Study Guide to Accompany Shari Lawrence Pfleeger's

Software Engineering: Theory and Practice

By
Forrest Shull and Roseanne Tesoriero

Table of Contents

Course Summary

This course is organized so as to, first, provide a general introduction to software development and identify the important phases of any software project. Then, each of the phases is examined in detail, in order to give the reader a picture of the current state of our understanding of software development.

Chapter 1 provides a general introduction to the field in order to give some sense of the magnitude and importance of software in today's world, the kinds of problems that make software development difficult, and an outline of how software development is undertaken. Chapter 2 provides more detail on the idea of a “software process”, that is, on the various stages software goes through, from the planning stages to its delivery to the customer and beyond. Different models of the process are introduced, and the types of project features for which each is most appropriate are discussed.

Chapters 3 through 10 follow, in order, the major phases in the life of a software system. Chapter 3 deals with the planning stages: how resources and cost are estimated, how risks are identified and planned, and how schedules are created. Chapter 4 details how the problem to be solved by the system (not the system itself) is defined. This chapter concentrates on the methods that are necessary to fully capture the customer's requirements for the system, and how to specify them in a way that will be useful for future needs. Once the problem is sufficiently well understood, the system that solves it can be designed. Chapter 5 discusses the design of the software, introducing broad architectural styles that may be useful for different types of systems as well as more specific design characteristics. This chapter sketches the roles of the people involved in producing the design, as well as measures that can be used to assess a design's quality. Chapter 6 explores an important design paradigm, Object-Orientation, in more detail and shows how the design notation captures useful information about several aspects of the problem and the resulting system. Chapter 7 discusses the general principles by which a system design is turned into working code. Chapters 8 and 9 discuss testing, an important activity for ensuring the quality of the code, in some detail. An overview of different types of testing, as well as testing tools and methods, are presented. Finally, Chapter 10 describes different types of training and documentation and what should happen when the system is delivered to the customer.

For many systems the responsibility of the developers does not stop at delivery. Chapter 11 discusses system maintenance, that is, the part of the life-cycle that comes after delivery. The nature of the problems that may arise with the system in this p hase, as well as techniques and tools for performing maintenance, are presented. Special emphasis is placed on what can be done during system development to minimize the effort required during maintenance.

Having presented a wide array of tools and techniques that can be used during the software process, the course next presents some guidelines for how an effective set of tools can be selected. Key to this idea of process improvement is the concept of empirically evaluating the different tools available. Chapter 12 presents the basic concepts behind empirical evaluation, including the different types of empirical studies. More specific guidelines are presented for evaluations of products, processes, and resources. Chapter 13 further illustrates this discussion by presenting specific process improvement examples in each of these categories. Chapter 14 examines what progress has been made in better understanding software development and the consequences of development decisions, and presents some closing thoughts on important future directions for software engineering.

Course Learning Objectives

This course should help you understand:

- What is encompassed by the field of study within computer science known as "software engineering." Your understanding of this field should include its past contributions, a sense of what is understood today about software development, and an overview of important and promising areas of future research.
- What it means to be a software engineer:
    - What kinds of activities are necessary for the production of a software system;
    - What the relationship with the customer should be like, and when to involve the customer in the software development process to ensure that the system meets his or her needs;
    - What the relationship with other members of the development team should be like, in order to achieve the complex, collaborative tasks that are necessary for developing large systems.
- What it means to be a software engineering researcher:
    - What kind of working relationship is needed with practitioners;
    - What types of research problems are of interest to researchers, and stand to give practical benefit to practitioners;
    - A general idea of how software engineering research is done.
- What is meant by a "software life-cycle":
    - What the important phases of software development are, and why each is necessary;
    - What types of intermediate products are produced in each phase;
    - How the phases relate to each other and to the finished product;
    - What type of activities a software engineer must complete in each phase.
- Particular techniques and tools that have been applied to software development, and the circumstances under which they may be more or less appropriate.
- How software projects are planned and managed:
    - What types of resources are involved in software development projects;
    - How risks are identified and assessed;
    - How predictions and assessments are made.
- How software process improvement can be achieved. You should also have an understanding of the role of empirical studies in process improvement, including the general types of empirical studies and the kinds of answers each is able to give to software problems.

Chapter 1: Why Software Engineering?

Learning objectives:
After studying this chapter, you should be able to:

- Define what is meant by software engineering and describe the differences between computer science and software engineering.
- Understand the track record of software engineering.
- Identify the characteristics of “good software”.
- Define what is meant by a systems approach to building software and understand why a systems approach is important.
- Describe how software engineering has changed since the 1970s.

Summary:
This chapter addresses the track record of software engineering, motivating the reader and highlighting key issues that are examined in later chapters. In particular, the chapter uses Wasserman's key factors to help define software engineering. The chapter also describes the differences between computer science and software engineering and explains some of the major types of problems that can be encountered. The chapter explores the need to take a systems approach to building software. The main emphasis of this chapter is to lay the groundwork for the rest of the book.

Software engineers use their knowledge of computers and computing to help solve problems. For problem-solving, software engineering makes use of *analysis* and *synthesis*. Software engineers begin investigating a problem by analyzing it, breaking it into pieces that are easier to deal with and understand. Once a problem is analyzed, a solution is synthesized based on the analysis of the pieces. To help solve problems, software engineers employ a variety of methods, tools, procedures and paradigms.

To understand where software engineering fits in, it is helpful to consider the field of chemistry and its use to solve problems. A chemist investigates various aspects of chemicals while a chemical engineer applies the chemists' results to a variety of problems. In a similar manner, computer scientists provide the theories and results that are used by software engineers to solve problems.

The development of software involves requirements analysis, design, implementation, testing, configuration management, quality assurance and more. Software engineers must select a development process that is appropriate for the team size, risk level and application domain. Tools that are well-integrated and support the type of communication the project demands must be selected. Measurements and supporting tools should be used to supply as much visibility and understanding as possible.

Software engineering has had both positive and negative results in the past. Existing software has enabled us to perform tasks more quickly and effectively than ever before. In addition, software has enabled us to do things never done before. However, software is not without its problems. Often software systems function, but not exactly as expected. In some cases, when a system fails, it is a minor annoyance. In other cases, system failures can be life-threatening. This has led software engineers to find methods to assure that their products are of acceptable quality and utility. Quality must be viewed from several different perspectives. Software engineers must understand that technical quality and business quality may be very different.

Exercises:

1. What is software engineering and how does it fit into computer science?
2. What is the difference between technical and business quality? Explain why each is important.
3. Give two or three examples of failures you have encountered while using software. Describe how these failures affected the quality of the software product.
4. Examine failures that have occurred in software that you have written. Identify and list the faults and errors that caused each failure.
5. Look through several issues of software magazines (IEEE Computer and IEEE Software are good choices) from the 1970's, 1980's and recent issues. Compare the types of problems and solutions described in the older issues with those described in the more recent issues.

Answer Guidelines:

1. To answer this question, you may find it useful to re-read Section 1.1. Software engineering is the study or practice of using computers and computing technology to solve real-world problems. Computer scientists study the structure, interactions and theory of computers and their functions. Software engineering is a part of computer science in that software engineers use the results of studies to build tools and techniques to meet the needs of customers.

2. Technical quality emphasizes the technical performance of a software product. Often, it is measured by the number of faults, failures and timing problems. Business quality focuses on the value of the software product for the business. It is measured by return on investment (ROI). ROI may be viewed very differently depending on the organization. In the Brodman and Johnson (1995) study, different views of ROI were found with the U.S. government and U.S. industry. The U.S. government views ROI in terms of dollars saved while U.S. industry views ROI in terms of effort savings.

   Both technical and business quality are important. A software product may have technical quality in that it performs the way it is intended or specified to perform. But, if the software system is not used for business functions, the system is not providing value to the business. In this case, the system would have technical quality, but not business quality. Similarly, a software product can provide functionality that is vital to the business, yet the technical quality may be poor. Ideally, a software product should have both technical and business quality.

   You may find it useful to re-read Section 1.3.

3. Answers to this question will vary depending upon your experiences. In your answer, you should include the following:

   - A description of the failure. Explain how the system performed in a way that was different from its required behavior.
   - A list of quality characteristics that have been violated by the failure.

You may find it useful to use McCall's quality model (from Figure 1.5 of the textbook) as a checklist. That is, use the items listed in the model to ask questions about the failures you describe. For example, was the failure that you experienced related to correctness? Were the results incomplete or inconsistent? Did the failure affect the system's usability? These are a sample of the questions that you may want to consider.

4. Answers will be specific to the types of failures that you identify. The purpose of this exercise is to make the distinction between errors, faults and failures clear. Review the definitions for errors, faults and failures. These definitions can be found in Sidebar 1.1 of the textbook.

5. Answers to this question will vary depending upon which articles are involved. To answer this question, you may want to use the seven key factors that have altered software engineering (from Wasserman (1996) and presented in Section 1.8 of the textbook) to make your comparison among articles from the past and recent articles. In your comparison, cite specific examples of how the problems and solutions have changed.

Chapter 2: Modeling the Process and Life-Cycle

Learning Objectives:
After studying this chapter, you should be able to:
- Define what is meant by the term “process” and how it applies to software development.
- Describe the activities, resources and products involved in the software development process.
- Describe several different models of the software development process and understand their drawbacks and when they are applicable.
- Describe the characteristics of several different tools and techniques for process modeling.

Summary:
This chapter presents an overview of different types of process and life- cycle models. It also describes several modeling techniques and tools. The chapter examines a variety of software development process models to demonstrate how organizing process activities can make development more effective.

A process is a series of steps involving activities, constraints and resources that produce an intended output of some kind. A process usually involves a set of tools and techniques. Processes are important because they impose consistency and structure on a set of activities. The process structure guides actions by allowing software engineers to examine, understand, control, and improve the activities that comprise the software process. In software development, it is important to follow a software development process in order to understand, control and improve what happens as software products are built for customers.

Each stage of software development is itself a process (or a collection of processes) that can be described by a set of activities. A process can be described in a variety of ways, using text, pictures or a combination. In the software engineering literature, descriptions of process models are prescriptions (or the way software development should progress) or descriptions (the way software development is done in actuality). In theory, the two should be the same, but in practice, they are not. Building a process model and discussing its subprocesses helps the team to understand the gap between the two.

Every software development process model includes system requirements as input and a delivered product as output. Some of the more common models include the waterfall model, the V model, the spiral model and various prototyping models. The waterfall model was one of the first models to be proposed. The waterfall model presents a very high-level view of what goes on during development and suggests the sequence of events a developer should expect to encounter. The V model is a variation of the waterfall model that demonstrates how testing activities are related to analysis and design. The spiral model combines development activities with risk management. No matter what process model is used, many activities are common to all.

There are many choices for modeling tools and techniques. There are two major categories of model types: static and dynamic. A static model depicts the process, showing that the inputs are transformed to outputs. A dynamic model can enact the process, so that the user can see how intermediate and final

products are transformed over time. The Lai notation is an example of a static modeling notation. The systems dynamics approach has also been applied to dynamically model software development processes.

Exercises:

1. Describe the process you use to get to ready for class or work in the morning. Draw a diagram to capture the process.
2. Describe three software development life-cycle models. For each, name the main activities performed, and the inputs and outputs of each activity. For each give an example of the kind of software development project where the life-cycle model would be well-suited, and an example of where the life-cycle model would be inappropriate; explain why.
3. What is the difference between static and dynamic modeling? Explain how each type of modeling is useful.
4. Use the five desirable properties of process modeling tools and techniques identified by Curtis, Kellner and Over(1992) and presented in Section 2.4 of the textbook to evaluate one process modeling tool or technique. You may use an example from the book and/or consult outside sources.
5. Explain the difference between prescriptive and descriptive process models. What is the purpose for each? When is it appropriate to use each?

Answer Guidelines:

1. When answering this question, consider the definition of a process. Your answer should include the following:
    - the activities involved
    - the steps required to complete the tasks
    - the inputs and outputs to each activity
    - the constraints involved

   You may find it useful to re-read Section 2.1.

2. Answers to this question will vary depending upon the life-cycle models chosen. Section 2.2 describes several life-cycle process models. Your answer should include the activities, the inputs and the outputs involved with each process model. In addition, you should provide examples and reasons why a particular process model would be appropriate as well as situations where a process model would be inappropriate. For example, if a development project is highly risky (development team is inexperienced with the domain, time pressures exist) a spiral life-cycle model would be appropriate because development activities are combined with risk management to minimize and control risk. However, if the development project is low risk, a spiral model may not be the best choice.

3. A static process model describes the elements of a process. It depicts where the inputs are transformed to outputs. A dynamic process model enacts the process and allows the user to view how the products are transformed over time. A static model is useful to identify the elements of the process. A dynamic model may be useful to simulate how changes to the process affect the outputs of the process over time.

   For more details on static and dynamic process models, re-read Section 2.3.

4. Your answer to this question will depend upon which process modeling technique or tool is chosen. Your answer should address the five desirable properties of process modeling tools and techniques outlined in

Section 2.4. Does the tool or technique you are evaluating possess the desirable characteristic? Which features of the tool or technique satisfy the desirable property? Are there areas where the tool or technique lacks support for a desirable property?

5. Descriptive models attempt to describe what is actually happening in the process. Prescriptive process models attempt to describe what should be happening with the process. For more details on prescriptive and descriptive process models, you may find it helpful to re-read Section 2.2. In your answer to this question, use the reasons for modeling a process (in Section 2.2) to describe how and when prescriptive and descriptive models are useful. Can you think of cases where a prescriptive process model may be inappropriate? How does a descriptive model help in building a prescriptive model?

Chapter 3: Planning and Managing the Project

Learning Objectives:
After studying this chapter, you should be able to:

- Understand how to track project progress.
- Identify different communication styles of personnel and how these styles affect team organization.
- Apply several effort and schedule estimation models.
- Identify risks and understand what is meant by risk management.
- Describe how process models and project management fit together.

Summary:
This chapter looks at project planning and scheduling by examining the activities necessary to plan and manage a software development project. It introduces some of the key concepts in project management, including project planning, cost and schedule estimation, risk management, and team organization. The chapter introduces notations that support project management activities. It also presents several examples of estimation models used to estimate cost and size.

The software development cycle includes many steps, some of which are repeated until the system is complete and the customers and users are satisfied. However, before committing funds for a software development or maintenance project, a customer usually wants an estimate of how much the project will cost and how long the project will take.

A project schedule describes the software development cycle for a particular project by enumerating the phases or stages of a project and breaking each into discrete tasks or activities to be done. The schedule is a time-line that shows when activities will begin and end, and when the related development products will be ready.

A systems approach of analyzing and synthesizing can be used to determine a project schedule. In the analysis of a project, a clear distinction between milestones and activities must be made. An activity is a part of the project that takes place over a period of time, whereas a milestone is the completion of an activity--a particular point in time. An analytical breakdown of the project into phases, steps and activities gives software engineers and the customers an idea of what is involved in building and maintaining a system. The analytical breakdown of the project is sometimes referred to as the work breakdown structure. From the work breakdown structure, an activity graph depicting the dependencies can be drawn. To make the activity graph more useful, the estimated time to complete each activity can be added to the graph. Then, the critical path method (CPM) can be used to determine the minimum amount of time it will take to complete the project, given the estimates of each activity's duration. In addition, the CPM reveals those activities that are most critical to completing the project on time. There are many tools available to support the tracking of a project's progress.

The number of people that will be working on the project, the tasks they will perform, and the abilities and experience they must have to do their jobs effectively are all factors that are used to determine a project schedule and estimate the associated effort and costs. As the number of people on a project increases, the number of possible lines of communication grows quickly.

Breakdowns in communication can affect a project's progress. The degree of communication and the work styles of project team members should be considered when deciding on the organizational structure of the team. There are several choices for team structure, from a hierarchical chief programmer team to a loose, egoless approach. Each has its benefits, and the appropriateness of each depends to some degree on the uncertainty and size of the project.

One of the crucial aspects of project planning and management is understanding how much the project is likely to cost. Cost estimation should be done early and often, including input from team members about progress in specifying, designing, coding and testing the system. To address the need for producing accurate estimates, software engineers have developed techniques for capturing the relationships among effort and staff characteristics, project requirements, and other factors that can affect the time effort and cost of developing a software system. Many effort-estimation methods rely on expert judgment, estimates based on a manager's experience with similar projects. Algorithmic methods are based on data from past projects. With algorithmic methods, models that express the relationship between effort and the factors that influence it are generated. The models are usually described using equations, where effort is the dependent variable, and several factors (such as size, experience, and application type) are the independent variables. Most of these models acknowledge that project size is the most influential factor. Machine learning methods are another alternative to expert judgment and algorithmic methods.

Project managers take steps to ensure that their projects are done on time and within effort and cost constraints. Managers must also determine whether any unwelcome events may occur during development or maintenance, and make plans to avoid these events or, if they are inevitable, minimize their negative consequences. A risk is an unwanted event that has negative consequences. Project managers engage in risk management to understand and control risks in a project. As with cost estimation, the project team can work to anticipate and reduce risk from the project's beginning. Redundant functionality, team reviews, and other techniques can help the team catch errors early, before they become embedded in the code as faults waiting to cause failures. Cost estimation and risk management can work hand in hand; as cost estimates raise concerns about finishing on time and within budget, risk management techniques can be used to mitigate or even eliminate risks.

Exercises:

1. Describe the process of getting a degree (bachelor's, master's or PhD) as a work breakdown structure. Draw an activity graph for the process. What is the critical path?
2. Describe the organizational structure for your work environment. Classify the working styles of several of your co-workers. What are the advantages to this structure? Do you see any problems with the current structure?
3. Discuss two techniques for making a prediction for effort. In particular, explain where during the development process the prediction is made, and when (if at all) the prediction is repeated.
4. Any prediction generates an estimate, E, that can be compared eventually to an actual value, A. Name two values that can be calculated from E and A to help determine the accuracy of the estimating process. Define the two values and discuss how the values for each are used to tell us that a prediction is acceptable.
5. Describe two different size measures and the advantages and disadvantages of using each.

Answer Guidelines:

1. The answer to this question will depend on the degree chosen and the process involved. Your work breakdown structure should include phases, steps, activities and milestones. You must also consider constraints such as time limits and pre-requisites. For example, to get a PhD, you may have to complete coursework, pass comprehensive exams, pass a preliminary exam, and defend your dissertation. There may be constraints, such as a two year time limit to complete all coursework or all comprehensive exams must be successfully completed before a preliminary exam can occur.

   For more details on work breakdown structures, activity graphs and critical paths, re-read Section 3.1.

2. Your answer will depend on your work environment. Use the descriptions of organizational structures to characterize your work environment. In Section 3.2, chief programmer teams are described. Section 3.6 describes several management structures such as matrix organizations and integrated product development teams. When evaluating the structure, you should consider whether the environment is highly or loosely structured. You should consider the number of potential lines of communication.

   When describing the working styles of your co-workers, keep in mind the styles described in the chapter: rational introverts, rational extroverts, intuitive introverts, and intuitive extroverts. Consider how these working styles affect communication in your work environment.

3. The chapter covers several different techniques for predicting effort. Algorithmic models and machine-learning models are presented in Section 3.3. Project planning is covered in Section 3.8.

4. Two measures of an estimate's accuracy are the mean magnitude of relative error (MMRE) and the percent of projects with estimate values within x percent of the actual value (PRED). The MMRE is the average of $|E-A|/A$ for each project. The PRED(x) is n/N where n is the number of projects with $|E-A|/A < x$ and N is the total number of projects. When the MMRE < 0.25, the technique is considered fairly good. Some researchers would like the MMRE to be less than 10%. For the PRED, a PRED(0.25) > 0.75 is considered good. The PRED(0.25) criterion means that 75% of the project estimates were within 25% of the actual values.

5. When estimating effort, it is often necessary to estimate size. Examples of size measures include lines of code, function points and object points. Object points can be calculated early in the process, but object points are a coarse measurement. Lines of code are not available early in the process, but are relatively easy to calculate. Function points can be calculated earlier than lines of code and provide a richer system description than object points.

Review Exam 1

1. If a system is being developed where the customers are not sure of what they want, the requirements are often poorly defined. Which of the following would be an appropriate process model for this type of development?
    a. prototyping
    b. waterfall
    c. V-model
    d. spiral

2. The project team developing a new system is experienced in the domain. Although the new project is fairly large, it is not expected to vary much from applications that have been developed by this team in the past. Which process model would be appropriate for this type of development?
    a. prototyping
    b. waterfall
    c. V-model
    d. spiral

3. Which of the following are potential barriers to the consumer of a reusable component?
    a. It is unclear where the responsibility for component failures lies.
    b. Sometimes, it takes more time to find a reuseable component than it would to build it.
    c. It can be costly to understand the intended behavior of a reuseable component.
    d. a and b only
    e. b and c only
    f. a and c only
    g. a, b and c

Suppose a library system is being developed. The system has three major subsystems: one that handles the check-out/check-in transactions; one that handles inventories; and one that handles reports. During the development of the system, several problems occur. Identify the problems as errors, faults or failures.

4. In the code for calculating late fees, the *fine_total* variable is not initialized.
5. While a librarian is attempting to add a new book title to the inventory, the system shuts down.
6. The requirements writer is unaware that a library card is not necessary for the check-in transaction.
7. In the requirements document, a late fee is specified as $0.25 per day with a maximum of $15. The code for calculating the late fee does not check for the maximum fee.
8. Every evening at 11pm, the library system is supposed to perform a backup of the daily transactions. The backup for Tuesday night did not occur.

9. Paul, a manager of the development team, decides to use a COTS product developed by Reports 2 U, a third party vendor, as part of the inventory subsystem. Which of the following are valid concerns:
    a. The COTS product may no longer be supported by the vendor at some later date.

   b. In order for the COTS product to work with the new system, a modification or enhancement to the COTS product may be needed. The vendor may be unwilling to make the change.
   c. The COTS product may not function as specified.
   d. a and b only
   e. b and c only
   f. a and c only
   g. a, b and c

Jenna, a project manager, has developed a new technique for estimating project size. She has been using the new technique on several projects. Her estimates and the actual values for project size are shown below. The criteria for a good estimating technique are: 75% of the estimates should be within 25% of the actual; and the mean magnitude of the relative estimate errors should be less than 25%. Use the table of project size estimates and the criteria given to answer the questions about Jenna's estimating technique.

Project	Estimate	Actual
A	8060	8000
B	9000	10000
C	7000	7200
D	15000	13000
E	10000	9600

10. Given the table of estimates and actuals, what is the MMRE? Round to the nearest 1/100.
    a. 0.01
    b. 0.05
    c. 0.06
    d. 0.07
    e. 0.10

11. What is the PRED(.25)?
    a. 0.05
    b. 0.25
    c. 0.33
    d. 0.75
    e. 1.00

12. Based on the criteria for a good estimation technique and the estimate data gathered so far, is the new technique a good one? (Yes/No)

Suppose Madeline, Andrew and Jason are three managers asked to estimate effort required to build a 50,000 lines of code project. Each manager uses a different estimating technique.

13. Madeline uses the basic, Walston/Felix model. What will her estimate (in person-months) be? Round to the closest month.
    a. 185 person-months
    b. 572 person-months
    c. 620 person-months
    d. 79634 person-months
    e. 99134 person-months

14. Andrew uses the Bailey and Basili basic model. What will his estimate (in person-months) be? Round to the closest month.
    a. 65 person-months
    b. 74 person-months
    c. 1189 person-months
    d. 1246 person-months
    e. 206129 person-months

15. Jason uses expert judgment to arrive at a 400 person-month estimate for the project. Using the estimates of Madeline, Andrew and Jason, what is the Delphi estimate for this project? Round to the closest month.
    a. 220 person-months
    b. 400 person-months
    c. 720 person-months
    d. 755 person-months
    e. 101888 person-months

16. If Madeline's estimate is used and there are 12 team members working on the project, how many months will the project take? Assume all team members can work concurrently. Round to the closest month.
    a. 15
    b. 48
    c. 52
    d. 6636
    e. 8261

17. Answer TRUE or FALSE:
    a. A development project is just beginning. An initial prototype of the user interface has been completed. It would be appropriate to use the COCOMO 2.0 Stage 1 at this point in the development.
    b. A design has been chosen and development has begun. Detailed information about the design is known. The COCOMO 2.0 stage 2 model would be appropriate at this point in the development.

18. System A has 4 screens and 3 reports. Of the 4 screens, 3 are medium and 1 is difficult. Of the reports, 2 are medium and 1 is difficult. System B also has 4 screens and 3 reports. For system B, 2 screens are medium and 2 are difficult. The 3 reports for System B are medium difficulty. Which system has more new object points (COCOMO 2.0, stage 1 model)?
    a. System A
    b. System B
    c. System A and B have the same number of new object points.
    d. It is impossible to determine from the information given.

Consider the following descriptions of different employees' work styles.

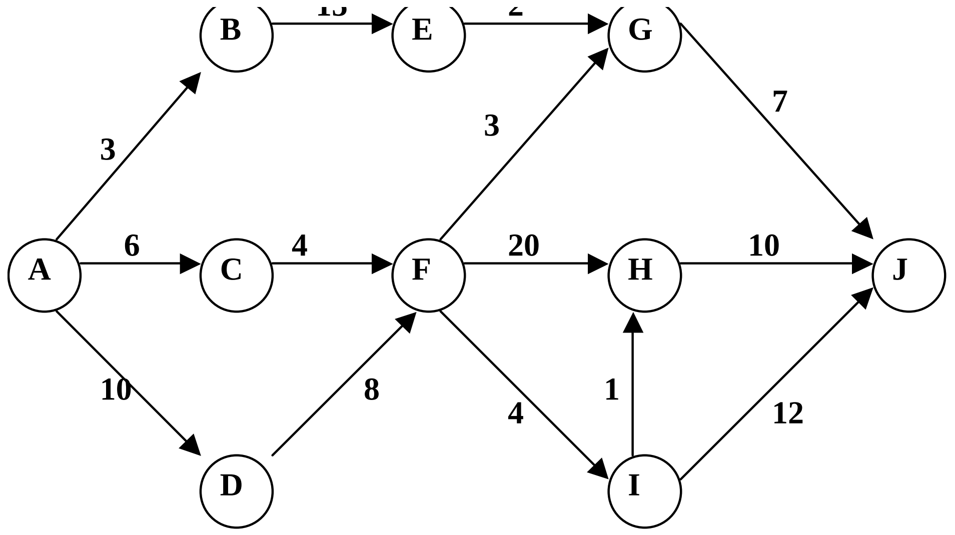
19. Kristie seeks out evidence to support her decisions. She is currently considering rearranging the office space to make the working environment more comfortable for the members of her team. While carefully considering the objective aspects of the change are important to her, she is also concerned about the opinions of the people who work for her. The members of Kristie's team consider her to be a good listener and often consult her when they have problems. Kristie's work style is:
    a. rational extrovert
    b. rational introvert
    c. intuitive extrovert
    d. intuitive introvert

20. Shane is an efficient leader. He knows what he wants and relies on his own experiences and logic to make decisions. He does not feel the need for extensive information before making a decision. He is capable of making fast decisions. Shane's work style would be described best as:
    a. rational extrovert
    b. rational introvert
    c. intuitive extrovert
    d. intuitive introvert

21. Jessica is a developer who enjoys trying new technology. She often finds inventive ways of incorporating new tools and techniques into the development process. After trying a new design tool, she immediately forms a positive opinion of the tool and attempts to get others to use the tool. Jessica's work style could best be described as:
    a. rational extrovert
    b. rational introvert
    c. intuitive extrovert
    d. intuitive introvert

22. Matthew is considering a new process for code reviews. He carefully seeks and reviews evidence to determine the potential benefits. He prides himself on being accurate and thorough. Matthew rarely looks to others for opinions. He would rather rely on information that can be objectively observed. Matthew's work style can be described as:
    a. rational extrovert
    b. rational introvert
    c. intuitive extrovert
    d. intuitive introvert

Activity graphs are used to depict the dependencies among the activities and milestones of a project. The nodes of the graph represent the milestones of the project. The edges linking the nodes represent the activities. The numbers adjacent to the edges represent the number of days required for the activity. For example, in the activity graph below, it will take 6 days to complete the activity starting at milestone A and ending in milestone C. Use this activity graph to answer the following questions:}



23. Which of the following is a critical path from milestone A to milestone J?
    a. ACFHJ
    b. ACFIHJ
    c. ABEGHJ
    d. ADFHJ

24. What is the slack time for the activity starting at milestone C?
    a. 7
    b. 8
    c. 15
    d. 20

25. What is the length of the critical path identified in question 23?
    a. 32
    b. 40
    c. 48
    d. 55

26. What is the latest start time for the activity starting at milestone E?
    a. 10
    b. 18
    c. 25
    d. 40

27. What is the earliest start time for the activities starting at milestone F?
    a. 11
    b. 19
    c. 33
    d. 37

28. Which milestones are precursors to H?
    a. A
    b. B
    c. C
    d. A and B
    e. A and C
    f. All of the above

29. If there are seven team members assigned to a project team, how many potential lines of communication are there?
    a. 6
    b. 7

c. 21
d. 49

Determine whether or not each of the following statements is describing a risk. Answer TRUE if the statement describes a risk, FALSE otherwise.

30. To catch defects early, requirements inspections have been incorporated into the process.

31. The customers are not clear about what they want. The requirements may be volatile.

32. The delivery of a subsystem being developed by another group may be delayed and cause the whole project schedule to slip.

33. The project team is inexperienced. A requirement may be misunderstood and designed incorrectly.

34. The development team is using a CASE tool for the first time on the design.

35. To aid the customer in identifying requirements, several prototypes are planned.

Review Exam 1 Answers

1. a, prototyping [Section 2.2]

2. b, waterfall [Section 2.2]

3. g; [Section 1.8]

4. fault [Sidebar 1.1]

5. failure [Sidebar 1.1]

6. error [Sidebar 1.1]

7. fault [Sidebar 1.1]

8. failure [Sidebar 1.1]

9. g; (COTS concerns)

10. d; MMRE = ((60/8000) + (1000/10000) + (200/7200) + (2000/13000) + (400/9600)) / 5 = 0.07 [Section 3.3]

11. e; All estimates are within 25% of actual values. [Section 3.3]

12. Yes; using criteria MMRE < 0.25 and PRED(0.25) > 0.75. [Section 3.3]

13. a; 185 person-months (Walston/Felix) [Section 3.3]

14. b; 74 person-months (Bailey/Basili basic model) [Section 3.3]

15. a; 220 person-months is the average of the three estimates [Section 3.3]

16. a; 15 months (duration on Madeline's estimate) [Section 3.3]

17. COCOMO [Section 3.3]
    a. TRUE
    b. FALSE, stage 3 would be more appropriate than stage 2 because detailed information about the design is known.

18. a; difficulty of reports is weighted more heavily than difficulty of screens in the COCOMO 2.0 model. [Section 3.3]

19. d; intuitive introvert [Section 3.2]

20. a; rational extrovert [Section 3.2]

21. c; intuitive extrovert [Section 3.2]

22. b; rational introvert [Section 3.2]

The following table can be used to answer questions 23 to 28:

Activity	Earliest Start Time	Latest Start Time	Slack
A	1	1	0
B	4	25	21

C	7	15	8
D	11	11	0
E	19	40	21
F	19	19	0
G	22	42	0
H	39	39	0
I	23	37	14
J(finish)	48	48	0

An activity label in the table should be read, “the activity beginning at milestone <*label*>.” For example, the activity beginning at milestone B has an earliest start time of 4.

23.d; ADFHJ is the critical path [Section 3.1]

24.b; 8 is the slack time for the activity starting at milestone C. [Section 3.1]

25.c; 48 is the length of the critical path. [Section 3.1]

26.d; latest start time for the activity starting at milestone E is 40. [Section 3.1]

27.b; earliest start time for the activity starting at F is 19 [Section 3.1]

28.e; B is not a precursor to H [Section 3.1]

29.c; (n(n-1))/2= (7(6))/2 = 21 lines of communication [Section 3.2]

30.FALSE; This is a risk control. [Section 3.4]

31.TRUE; Requirements volatility is a risk. [Section 3.4]

32.TRUE; Late delivery is a risk. [Section 3.4]

33.TRUE; Team inexperience is a risk. [Section 3.4]

34.TRUE; First use of a new technology is a risk.

35.FALSE; Prototyping is a risk control. [Section 3.4]

Chapter 4: Capturing the Requirements

Learning Objectives:
After studying this chapter, you should be able to:

- Explain why it is necessary to elicit requirements from software customers, and the role of requirements in the software life-cycle;
- Identify the characteristics that make individual requirements good or bad;
- Describe the types of requirements that should be included in a requirements document;
- Describe the notations and methods that can be used for capturing requirements, and the types of situations in which each may be appropriate;
- Explain how and why requirements reviews should be done to ensure quality;
- Describe how to document requirements for use by the design and test teams.

Summary:
This chapter focuses on capturing system requirements, an important component of any model of the software development process. It is important to remember that the purpose of requirements is to specify the problem that the system is intended to solve, leaving the details of the solution to the system designers. Formulating a useful set of requirements will require working closely with:

- customers and users, so that everyone understands the requirements and their goals
- designers, so that they can construct a good design from the requirements specification
- testers, so that they can write test scripts to evaluate whether the implementation meets the requirements
- documentation writers, so that they can write user manuals from the specifications

Any requirements document should include both functional and non-functional requirements. The functional requirements explain what the system will do, and the non-functional ones constrain the behavior in terms of safety, reliability, budget, schedule and other issues. Since mistakes made during the requirements process can cause additional problems later in the software life-cycle, the complete set of requirements should be validated by checking for completeness, correctness, consistency, realism, and other attributes. Measures reflecting requirements quality are especially important since they may indicate useful activities; e.g. when indicators show that the requirements are not well-understood, prototyping of some requirements may be appropriate.

There are many different types of definition and specification techniques that can be used for capturing requirements. Some are static (e.g. data flow diagrams), while others are dynamic (i.e. they include information about timing and time-related dependencies). We can also think of techniques as object-oriented or procedural. The techniques that are used on a particular software development project should be chosen carefully, based on a number of factors. For example, the specification techniques differ in terms of their tool support, maturity, understandability, ease of use, and mathematical formality. Projects vary in terms of size and scope. The right technique must be chosen based on the needs of the current project, keeping these factors in mind. In some cases it may be desirable to use a combination of techniques to specify the different aspects of a system.

Because requirements typically contain many disparate elements that are integrated into a comprehensive whole, requirements must be written in a way that allows them to be linked and controlled. For example, a change to one requirement may affect other, related requirements, and the techniques and tools must support the changes to ensure that errors are caught early and quickly.

Exercises:

1. Most of a system's requirements specify that the system should do what it is intended to do. Is it also appropriate to specify that the system should *not* do what it is *not* intended to do? If your answer is no, explain why; if your answer is yes, give an example.
2. Describe the different consumers of software requirements i.e. the different users, or types of users, of a software requirements document). For each consumer, explain how he or she would use the requirements, and how the requirements should be documented to make them useful for this consumer.
3. One source of problems in the requirements phase can be the relationship between system developers and their customers. What are some negative stereotypes customers may hold about developers? What could you, as a developer on a project, do to minimize the impact of those negative stereotypes?
4. Download and read the Robertsons' requirements definition template (from http://www.atlsysguild.com). Write a short report in which you summarize for a software developer how he or she can use the template to validate requirements. Your report should address practical concerns about how to use the template in the requirements process. For example, you should address questions such as: At what point in the requirements process can the template be helpful? What activities are necessary in order to use the template for validating requirements? What skills will a developer have to possess in order to apply the template effectively? What types of errors and faults will the template help uncover?
5. Pamela Zave has proposed a classification scheme for organizing the different types of research that go on in the area of software requirements (P. Zave (1997). "Classification of research efforts in requirements engineering." ACM Computing Surveys, 29(4): 315-321). Choose one of the categories she presents and write a brief (one paragraph) description of how research in this area is of use to software developers. Track down one of the papers she cites as an example of research in this category, and summarize the problem it addresses and the results it presents.

Answer Guidelines:

1. An example of the latter type of requirement is a security requirement. In this case, it is necessary to specify exactly what the system should NOT allow a user or other system to do.
2. Requirements need to be used by:
   a. The customer, who should check that the system described actually matches his or her needs. For use by the customer, requirements should be easy to understand, with a minimum of jargon, to facilitate clear communication with the customer.
   b. Designers, who need to construct a design of the system described in the requirements. The requirements will need to be as complete, clear, and correct as possible so that designs developed from it are

correct. Also, they will need to identify all of the constraints on the system so that the design can correctly incorporate them.

c. Testers, who need to develop test scripts. To support testers the requirements should be as precise as possible, so that the values that need to be tested and the expected system behavior are well-specified.
d. Documentation writers, who will write the user manuals based on the requirements. As for the customer, the requirements should clearly communicate the features of the system.

3. Use Table 4.5 as a starting point for your answer. For each point listed on the table under the category of “How users see developers,” think about whether or not these prejudices are in fact true, and if so, if there is a reason they must be that way; if not, ask what you could do to change that perception. For example, the first point, “Developers don't understand operational needs,” is true in many cases; software engineers don't always have extensive training in the customer's application domain or way of doing business. But, a serious effort by the developer to learn about the customer's needs in order to support the customer is not only helpful for reversing negative stereotypes, it is a prerequisite for building quality software.

4. The template is useful during requirements reviews, but it can also be helpful if given to the writers of requirements as a guide for what information should be included. The categories and items of the template can be used as a checklist for ensuring that all of the appropriate issues have been addressed and the correct information has been included in the requirements document. The reviewer of the requirements will have to have a sufficient understanding of the requirements in order to find the pertinent information for each item of the template. The template will be most helpful for finding defects of omission, that is, for identifying types of information that should be included in the requirements but were left out.

5. Answers will vary depending on the category chosen and papers selected.

Chapter 5: Designing the System

Learning Objectives:
After studying this chapter, you should be able to:

- Explain the difference between a conceptual design and technical design, and the reasons why each is useful for software development;
- Describe an overview of important design styles, techniques and tools, and the conditions under which different choices may be appropriate;
- Identify the characteristics of a good design;
- Explain why validating designs is necessary, and a general overview of how this task can be accomplished;
- Explain how to usefully document a design.

Summary:
This chapter focuses on the process by which the requirements (the description of what the customers want the system to do) are translated into a design (a description of a system that will satisfy the customers' needs).

Of course, a good design is one that describes a system able to meet all of the requirements. However, other high-level concepts are important, too. For example, it is important that the design is adequate for the long-term intended use of the system and embodies the following high-level notions of quality:

- Reusability: Are components of this design likely to be reused in later systems? If so, are they of sufficient quality to be reused?
- Understandability: Is the design well structured and documented so that it will be easy for maintainers (who may or may not include software engineers who developed the system originally) to understand where in the system modifications need to be made?
- Modifiability: Will the system described in this design be easy enough to maintain after implementation is over? Or will changes likely have unintended consequences?

These are high-level concepts that try to describe design quality, but lower-level measures can be helpful as well. Concepts such as modularity, abstraction, coupling, and cohesion measure important characteristics of a design and allow the formulation of general guidelines. For example, in most cases a system with low coupling and high cohesion will be easier to understand and hence maintain than a system without those characteristics. While it is not possible to say for a given system exactly how much coupling or cohesion is appropriate, measurement of these values may be useful in evaluating the quality of components, or for predicting which components are likely to be costly to build or maintain.

Other important considerations for the system also need to be decided in the design phase, rather than during implementation. Since it is important that the system meets the customer's needs, it is appropriate for the designers to work with users to decide how to design the system's interface. This may require developing several prototypes to show users the possibilities, to determine how performance requirements can be met, or to evaluate the best “look and feel.” Another example of a decision to be made at design time is fault tolerance. One goal of the design should be to anticipate potential faults that may occur and design the system in ways that minimize disruption to the user.

This chapter also emphasizes that design is an activity that involves other developers. Other developers are an implicit factor in the design process, since the choice of design method depends on who will have to read and understand the design. Also, since designs are built from components, the interrelationships among components and data must be well-documented. Cross-referencing may be necessary to help explain which parts of the design affect what components and data. Other developers should also participate in design reviews, evaluating the design at several stages and making suggestions for improvement. For all of these reasons, it is essential that a design is documented clearly and completely, with discussions of the options faced and the decisions made.

Exercises:

1. The following statements describe modules in a (hypothetical) program. For each, decide whether the module is likely to have a high or low degree of cohesion. If cohesion is low, explain why.
   a. Module "InventorySearchByID" searches the records in inventory to see if any match the specified range of ID numbers. A data structure is returned containing any matching records.
   b. Module "ProcessPurchase" removes the purchased product from inventory, prints a receipt for the customer and updates the log.
   c. Module "FindSet" processes the user's request, determines the set of items from inventory that match the request, and formats the items into a list that can be shown to the customer.
2. What does it mean to say that a design review should be "egoless"? Why are egoless reviews necessary? Suggest some steps that may help achieve egoless reviews.
3. Choose a software system that you use and for which there is some feature of the interface that you, as a user, dislike. Briefly describe the software and why you consider this feature of the interface to be a problem. Speculate as to whether the problem could have been avoided during design. If so, what changes to the design process would have been necessary?
4. Find a paper or book that deals with design patterns in more detail. Summarize briefly one of the patterns presented; explain what it is useful for, when it can be used, and its effects on the rest of the program. What are some of the difficulties you might expect to encounter if using design patterns in practice?
5. Read the account by Nancy Leveson and Clark Turner of the Therac-25 accidents (N. Leveson and C. Turner (1993). "An investigation of the Therac-25 accidents." IEEE Computer, 26(7) (July): 18-41). Give a brief summary of some of the important lessons that can be learned for design.

Answer Guidelines:

1.
   a. Module "InventorySearchByID" can be expected to have high cohesion; it performs only one type of functionality (a search).
   b. Module "ProcessPurchase" can be expected to have relatively low cohesion, since it involves very different functionalities: printing a receipt for use by the user is logically quite different from updating a data store.
   c. Module "FindSet"can be expected to have relatively low cohesion, since it invokes very different functionalities: parsing input, a search through data, and output formatting.
2. "Egoless" reviews occur when criticisms are directed at the design process and the design itself, not at the designers and other participants. Egoless reviews are useful since they remind participants that they are moving toward a common goal and keep them focused on the software under review, rather than encouraging them to make excuses or defend themselves.

Egoless reviews enhance communication and allow more time to be spent discussing the software itself. Egoless reviews can be facilitated by any measure that reminds participants that the software and not the individuals are under discussion. Examples include making a statement of the review goal at the beginning of the review, or making sure that comments during the review remain centered on the software. Egoless reviews are also facilitated by not inviting a representative of management, to make sure that participants do not feel their performance is being judged based on other participants' comments.

3. Answers will vary depending on software system chosen and interface problem discussed. In any case, you should first make sure that the problem you've selected is in fact a problem from the user's point of view (i.e. one that results from a poorly defined interface). For example, suppose you work with a piece of software that, at certain times, will not allow you to interact with the system while some computation is going on. The fact that you have to wait for the computation to complete may not be a user interface problem; the computation may simply require a large amount of time to complete the calculation. However, if the user is not given a chance to cancel the computation after it has started, then a frustrating problem can occur in which a user has decided that it is not necessary to run the computation at this point but has to wait for it to complete anyway. Many such problems can be caught in the design phase if resources are expended on user interface design and review.
4. The definition of a design pattern almost always includes context information, a definition of the pattern itself, and tradeoffs associated with its use, so be sure to address each of these points in your summary. Design patterns suffer from some of the same difficulties that are experienced in any reuse situation: it is often hard to recognize situations that could benefit from reuse, and it is difficult to search a repository of components (including patterns) to find the best match to the current situation. As with any other type of reuse, developers may not be sufficiently motivated to overcome these difficulties with design patterns.
5. The Leveson and Turner paper contains many lessons that can be applied to design. You should select some of these lessons, and cite experiences with the Therac-25 that support them. For example, page 39 contains a list of five basic software engineering principles that were violated in development of the Therac-25; which of these can be applied to design? What kind of conditions did their absence from the development of the Therac-25 lead to? You should use these principles as a starting point but also identify other lessons. For example, what kind of lessons can be found about reuse? About timing problems?

Chapter 6: Considering Objects

Learning Objectives:
After studying this chapter, you should be able to:

- Explain what is meant by object-oriented development, and how it differs from other development paradigms;
- Understand what use cases are, discuss why they can be useful in software development, and use them to describe system functionality;
- Use and understand UML diagrams;
- Explain how object-orientation is used for system design;
- Explain how object-orientation is used for program design;
- Explain how measurement is useful in object-oriented development, give examples of some object-oriented metrics, and explain the concepts those metrics are capturing.

Summary:
This chapter describes the Object-Oriented (OO) approach to development. Object-orientation organizes both the problem and solution according to several key concepts:

- Identity
- Abstraction
- Classification
- Encapsulation
- Inheritance
- Polymorphism
- Inheritance

Taken together, these concepts give OO development several advantages that other approaches do not possess. For example, OO does not impose a particular development process but can be used with many lifecycle models. Perhaps most importantly, OO allows a consistent terminology to be used across the stages of the lifecycle, so that each stage can build more directly on the analysis done in previous stages.

In OO development, first likely concepts and scenarios of system use are determined. Then, during design, corresponding classes and objects are identified, as are the interactions and relationships among them. This information is represented in a series of models that are translated to an OO programming language during the coding stage. Testing in OO development requires the same activities as non-OO testing (e.g. unit testing, system testing), although these activities are mapped to OO concepts such as classes and class hierarchies rather than functions and modules.

Use cases are a particularly useful way for representing system functionality, and are well-supported by an OO approach. Aside from describing information that is useful for many later stages of the development process, use cases are effective for communicating with customers, system designers, and testers.

OO solutions are often described using a notation support known as the Unified Modeling Language (UML). UML provides diagrams that capture information about the system in a series of dynamic and static views. The UML diagrams include workflow diagrams, object models, sequence diagrams, collaboration diagrams, package diagrams, component diagrams, and deployment diagrams.

An OO approach to design, like any other approach, generally proceeds through both system and program design phases. During system design, OO developers describe the problem at a high level of abstraction. They must identify classes and attributes in the system, find relations among the classes and understand their type (generalization, association, aggregation, or composition), and decide on the private or public interfaces of the classes.

In program design, OO developers expand the system design to describe the proposed system in more detail. They must decide on operation signatures and object interfaces, choose construction paradigms (e.g. black-box versus white-box reuse), and make decisions regarding user interfaces, data management, and task management in the system. At this stage of design, developers have a number of design aids from which to choose, such as toolkits, patterns, and frameworks.

For both system and program design, developers can use measurement to provide further insight into the design process. OO metrics can be broadly classified as either size or design metrics; many specific metrics have been defined in each category. There is no canonical best set of metrics; developers must consider what is useful and feasible to measure in their context before selecting which to use.

Exercises:

1. In what ways do the Object-Oriented characteristics of encapsulation and information hiding support reuse? What kind of criteria would you use in deciding whether to reuse a class in a new system?
2. Why is it useful to have separate phases for system and program design?
3. In section 6.3, a set of questions was given for helping to find potential problems with a set of use cases. Why is it important if two different terms are being used to refer to the same entity?
4. You are designing a system to help run a bookstore. Revenue for the store comes from two distinct services: customers can purchase books, or bring their books in for rebinding. You are considering making a separate class for each service, both of which would be subclasses of a general “sale item” class. What are the likely benefits of such an approach? Are there any possible arguments against using inheritance in this case? Be sure to specify what factors could influence your decision.
5. Section 6.7 discusses Chidamber and Kemerer’s metric of depth of inheritance. Why does it seem likely that a class that is deeper in the hierarchy is harder to understand and maintain than one that is less deep?

Answer Guidelines:

1. A good way to answer this question would be to turn it around and begin by thinking of what would make for an ideal reuse situation. Some goals could be: it should be easy for the developer to understand what functionality is available to be reused, related functionalities should be somehow reusable together, it should be easy to understand how to reuse the functionality, and the reusable components should be of high quality. Then, address whether it would be harder or easier to achieve these characteristics in an OO environment, and why.
2. System design gives developers a chance to solidify the broad outlines of the proposed system before having to decide on more specific details of the implementation. Use the beginning of section 6.6 to understand what decisions are reserved for program design. Then consider what negative

outcomes could result if developers began debating these issues earlier. What factors, later in development, could make the answers to these issues more or less relevant? Would it ever be a drawback to come to an early decision on these issues, even if the system could be implemented without changing these decisions in later lifecycle phases?

3. The point of UML diagrams and use cases is to serve as a basis for the eventual detailed design and implementation of the system. Using multiple terms for the same entity hinders using the use cases in this way because it can cause confusion for people reading the diagrams, perhaps leading them to believe that multiple, different entities are being referred to. In your answer, try to think of an example of this situation, and describe the likely results during system development in later lifecycle phases.
4. The likely benefits involve shared functionality, perhaps for pricing and revenue, which could be implemented once in the superclass and shared in each of these two classes. On the other hand, there are enough differences between the two services that it could be argued that joining such dissimilar classes in an inheritance hierarchy would be confusing, since very little functionality could be shared through the superclass. (For example, book sales might involve updating inventory, tracking the space required for sale, and ordering more copies of a book when it is close to being sold out. Bookbinding would not track inventory but would need another set of methods to deal with expected due dates for the service and scheduling the time of a suitable expert at the shop.) Many factors should be taken into account to make the decision, but in this case a particularly important one is the anticipated future needs of the store. If it is possible that the range of sale items might increase in the future, then it becomes more cost-effective to encapsulate the common functionality in a superclass for sharing among the classes that might be added.
5. To answer this question, think about how someone reading an OO program knows where the specific definition of a method is located. If there is no inheritance involved, a method is defined within the class to which it belongs. But if that class is part of an inheritance hierarchy, the method need not be defined in the class. Is there any indication of which parent class contains the definition of a method? Can a method be redefined multiple times within a particular hierarchy? How do those factors contribute to the ease and accuracy with which a method definition can be found?

Review Exam 2[1]

The following questions are in reference to a hypothetical “Gas Station Control System” (or GSCS) that will be used to help manage an American-style gasoline or service station. Our hypothetical gas station basically provides two services:

- There is a small store that carries car parts. Inside the store is at least one cash register, operated by a cashier who is an employee of the gas station.
- There are a number of gas pumps, at which customers can park their cars, interact with the system to pay via credit card, and then pump their own gas. Alternatively, the customer can pay for his or her gas via cash or credit card by going into the store and paying directly to the cashier.

Thus the GSCS has two main classes of users. The first is the cashier, who uses the GSCS to record purchases of car parts by customers. The GSCS must allow the cashier to enter the type and number of parts purchased, then compute the total purchase price and handle the payment. Customers purchasing gasoline are the second type of user. These customers interface with the system at the gas pump, by specifying the amount and type of gas they will buy, paying either at the pump or to the cashier, and then pumping the gas themselves.

The system also has to interact with other automated systems to perform its tasks. For example, in order to accept credit card payments, the GSCS must interface with a system maintained by the credit card company. The credit card system is responsible for checking that the customer's account is in good standing and can accommodate the amount of the purchase, and for debiting the customer's account and eventually reimbursing the gas station. The operation of these external systems is beyond the scope of the GSCS, although the GSCS needs to know how the external systems will communicate the success or failure of their tasks.

The first step that the development team decides to undertake is to create a requirements document to describe the system.

1. Which of the following statements best describe the benefits that the development team may expect from the requirements process?
   a. The requirements process can help team members understand how the different types of functionality in the system relate to each other.
   b. The requirements process can help the team make programming decisions, such as which is the best algorithm to use for computations, at an early point.
   c. The requirements process can help the team avoid omitting necessary functionality.
   d. a and b
   e. a and c
   f. b and c

[1] The design diagrams for the GSCS used in this exam are adapted from ones created by Prof. Guilherme Travassos, of the Federal University of Rio de Janeiro, and Jeffrey Carver, of the University of Maryland, College Park. The diagrams are reprinted from *Advances in Computers*, volume 54, Travassos, Shull, and Carver, “Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language,” 2001, by permission of the publisher Academic Press London.

   g. a, b, and c

2. Which of the following statements best describe the benefits that the owner of the gas station may expect?
   a. The requirements process can help the gas station owner think more clearly about the set of functionality that should be included in the GSCS.
   b. The requirements process can help the gas station owner and the development team in communicating while discussing the system.
   c. The requirements process gives the owner a specific statement of exactly what the final system will do.
   d. a and b
   e. a and c
   f. b and c
   g. a, b, and c

3. The development team needs to pick a representation for the requirements. Which of the following is a valid choice and rationale?
   a. Data flow diagrams, since major system functionality will involve data interfaces among hardware (e.g. cash register, pumps, credit card readers).
   b. Event tables, because the system will need to handle many events happening concurrently (e.g. multiple customers at multiple gas pumps).
   c. Z, because it is easily understandable by the gas station owner and will aid communication about the system.
   d. Object-oriented specification, since this will make sure that the system response is specified for every situation.

Mark the following TRUE if they belong in the requirements for the GSCS, and FALSE if they do not.

4. How much documentation the development team is required to produce.
5. The level of training that will be necessary for the cashiers to use the system effectively.
6. The constraint that new customers, paying for gasoline, must be able to learn how to use the system from simple directions posted at the gas pumps.
7. The maximum cost of the system.
8. The hardware constraints that are necessary for interfacing with the cash registers and gas pumps.
9. The format of the data received from the cash registers and gas pumps.
10. How maintenance to the system will be performed.

11. Which of the following excerpts could be considered valid requirements?
   a. "Once the payment process is complete, the system should respond in the following way: If the user has paid the cashier directly, or has paid at the pump but does not desire a receipt, then return to the initial state. Otherwise, print a receipt."
   b. "A record should be kept for each cashier. Each record should store the last name, first name, and employee ID number. The records should be maintained in a linked list."
   c. "After the user has selected a payment option, the system should check if the input is valid (i.e. a number between one and three)."
   d. a and b only
   e. a and c only
   f. b and c only

    g. a, b, and c

12. Which of the following are examples of valid nonfunctional requirements?
    a. “The display must update within three seconds after the user has selected a payment option.”
    b. “When car parts have been purchased, the count of inventory remaining must be updated. A warning message will be displayed if the count drops below the pre-set limit.”
    c. “The user must replace the nozzle when finished pumping gas.”
    d. a and b only
    e. a and c only
    f. a and c only
    g. a, b, and c

A requirements review is undertaken to make sure that the requirements adequately describe the system to be built.

In questions 13 through 17 , review the given excerpt from the requirements and decide whether it is an adequate requirement or not. If it should be rewritten, mark all the reasons that apply.

13. “After the payment process is complete, the relevant information should be appended to a log file.”
    a. This requirement should be rewritten; it is incorrect.
    b. This requirement should be rewritten; it is ambiguous or inconsistent.
    c. This requirement should be rewritten; it is unrealistic.
    d. This requirement should be rewritten; it is unverifiable.
    e. This requirement is fine.

14. “The system should be constructed so that it will be easy to add new functionality in the future.”
    a. This requirement should be rewritten; it is incorrect.
    b. This requirement should be rewritten; it is ambiguous or inconsistent.
    c. This requirement should be rewritten; it is unrealistic.
    d. This requirement should be rewritten; it is unverifiable.
    e. This requirement is fine.

15. “The price of a gasoline purchase is computed as the price per gallon for the type of gas purchased, multiplied by the number of gallons purchased (use two decimal points for representing fractions of gallons).”
    a. This requirement should be rewritten; it is incorrect.
    b. This requirement should be rewritten; it is ambiguous or inconsistent.
    c. This requirement should be rewritten; it is unrealistic.
    d. This requirement should be rewritten; it is unverifiable.
    e. This requirement is fine.

16. “The system should be easy for new customers to use.”
    a. This requirement should be rewritten; it is incorrect.
    b. This requirement should be rewritten; it is ambiguous or inconsistent.
    c. This requirement should be rewritten; it is unrealistic.
    d. This requirement should be rewritten; it is unverifiable.
    e. This requirement is fine.

17. “The system should be available 24 hours a day, 7 days a week.”
    a. This requirement should be rewritten; it is incorrect.
    b. This requirement should be rewritten; it is ambiguous or inconsistent.
    c. This requirement should be rewritten; it is unrealistic.
    d. This requirement should be rewritten; it is unverifiable.
    e. This requirement is fine.

After the requirements review, 23 changes are made and a new version of the requirements is created. This version was reviewed again, and 9 changes were suggested. After these were changed, the latest version was shown to the customer, who recommended 5 more changes.

18. Based on the above measurements, what can we conclude?
    a. The requirements should be reviewed again, since there are still defects being found.
    b. The development team should begin working on the design based on these requirements, since the number of changes is decreasing and the requirements are becoming more stable.
    c. It is impossible to say whether the requirements should be re-reviewed without knowing the types of changes being made.

The development team decides that the next step is to create a conceptual and then a technical design.

19. The conceptual design is felt to be of value because it will allow the gas station owner to check:
    a. What part of the system will be responsible for tracking the number of car parts left in stock, and how it communicates with other parts of the system.
    b. How the current price of a gallon of gasoline is input to the system.
    c. What the screens that the user sees will look like, and what options users will have.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

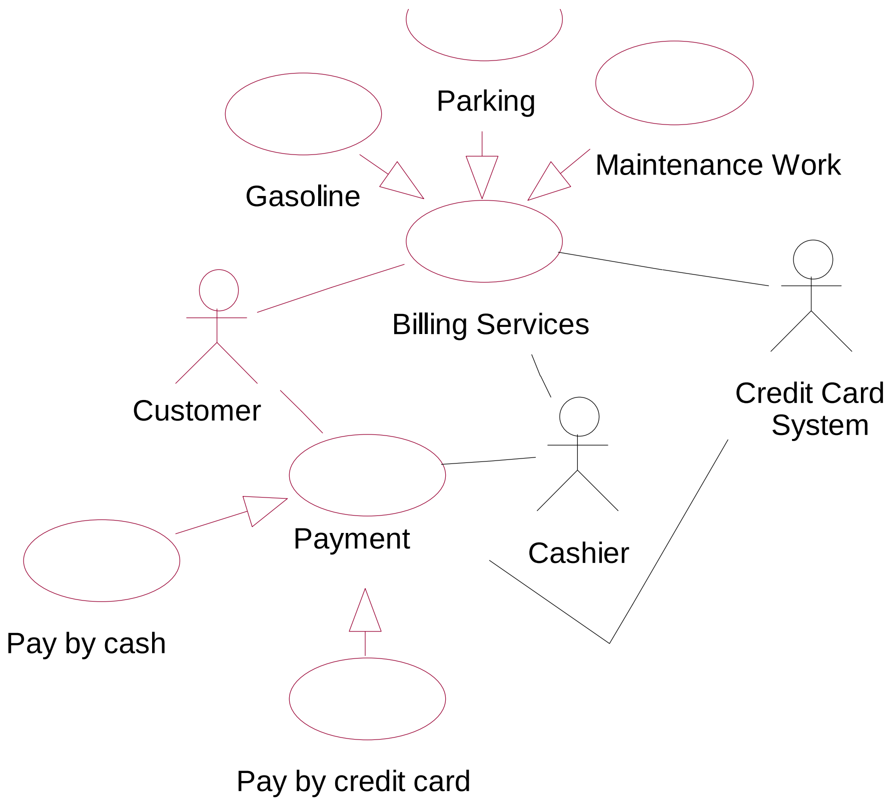
20. Which of the following are valid rationales for creating a separate technical design?
    a. The conceptual design will be useful for communicating with the gas station owner but not very useful as a basis for implementing the system.
    b. The technical design should contain more information about the gas pumps and their interfaces to the software.
    c. The technical design should contain more detail about the likely data structures that will be used.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

21. The team has to decide on a general approach to creating the design. Which are NOT valid choices and rationales?
    a. Modular decomposition, because the system can be divided into separate types of functionality that are relatively independent (for

example, the operation of the cashier versus the operation of the gas pumps).
   b. Outside-in design, because the set of user inputs is fairly well understood.
   c. Object-Oriented design, because the emphasis will mainly be on the flow of data through the system (for example, how the central system tracks purchases at each of the individual gas pumps).
   d. a and b
   e. a and c
   f. b and c
   g. a, b, and c

22. The team leader decides that the logical next step is to decide on an architectural style for the system. Which of the following are NOT valid choices and rationales?
    a. Object-Oriented, since the problem can be decomposed into several different entities, each responsible for its own data access and manipulation routines.
    b. Pipe and Filter, since most of the required functionality involves “piping” data between subsystems in preset ways.
    c. Implicit Invocations, since the system is event-driven and depends on the reliability of the subcomponents.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

23. TRUE or FALSE: Having decided on an architecture, the team leader decides that the architecture should be frozen for the life of the project. That is, once code design starts, no changes to the architecture will be permissible so that there will be no inconsistencies. This is a reasonable strategy.



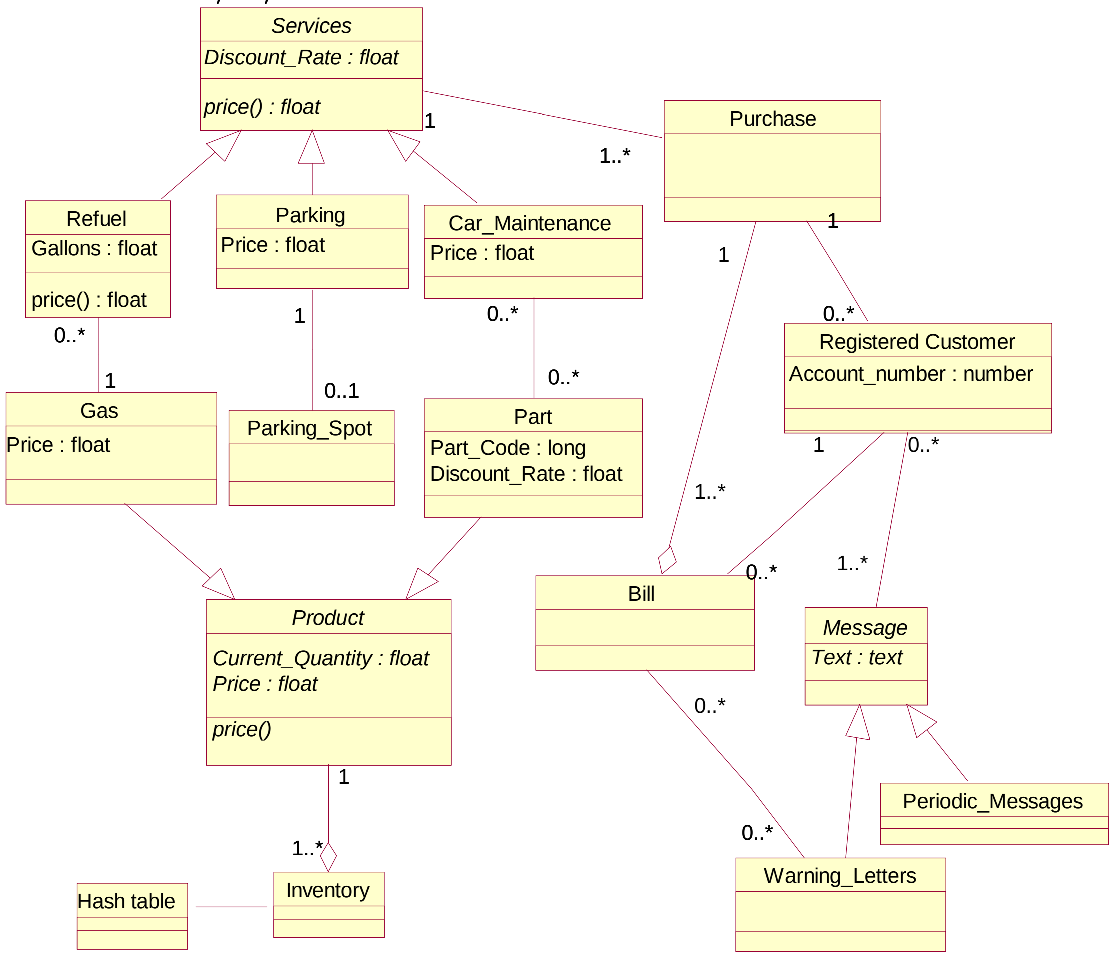
The team decides to use an Object-Oriented methodology to create the design. The figure above shows the first draft of the use case diagram for the gas station system, created during high-level (conceptual) design. Use it to answer questions 24 through 26.

24. Each of the ovals represents a particular high-level functionality of the system, and
  a. the lines between them represent the order in which they would typically be executed.
  b. a scenario should be constructed for each, to show the details of how the functionality would be supported by the system.
  c. each should have a specified start condition.
  d. A and B only.
  e. B and C only.
  f. A, B, and C

25. The notation of a triangle on the link between “Parking” and “Billing Services” signifies:
  a. The functionality described in “Parking” occurs before the functionality in “Billing Services.”
  b. The entities “Customer” and “Credit Card System” provide input to billing services, but are not involved with the system during parking.
  c. The functionality in “Parking” is a specific type of billing service.
  d. A and B only.
  e. B and C only.
  f. A, B, and C.

26. To check for any problems with the use cases, the team should:
  a. Review the customer’s description of the “Credit Card System” to see if it can participate in the appropriate way in the functionality described in billing services.
  b. Make sure that the expected start conditions for each use case are well understood.

   c. Combine “Cashier” and “Customer” into a single entity, since they are involved in the same set of use cases.
   d. A and B only.
   e. B and C only.
   f. A, B, and C



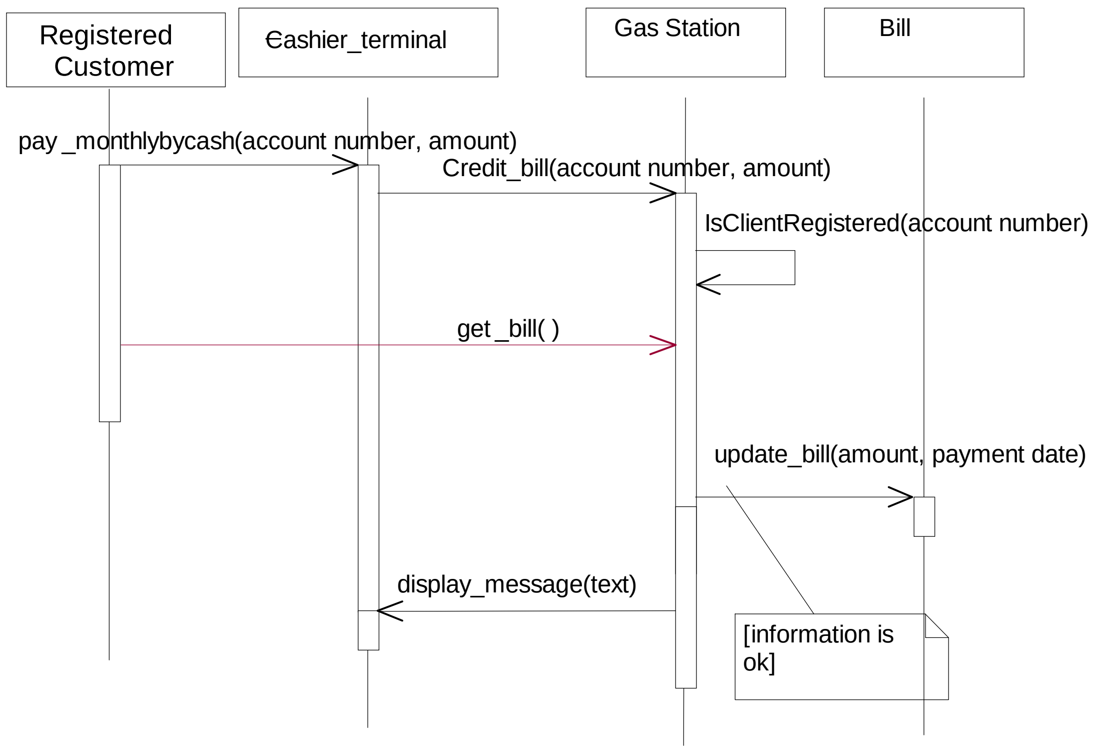
The figure above shows the first draft of the class diagram for the gas station system, created during high-level (conceptual) design. Use it to answer questions 27 through 29.

27.   The relationship between classes “Message” and “Registered customer” is that:
   a. A registered customer can have no associated message.
   b. Multiple messages can be associated with a registered customer.
   c. A message might exist, but be associated with no registered customer.
   d. A and B only.
   e. B and C only.
   f. A, B, and C.

28.   Which of the following statements are true about the GSCS, as described in this class diagram?
   a. Any subclass of “Service” must be associated with at least one instance of “Purchase.”
   b. A “Bill” includes exactly one “Purchase.”
   c. An instance of “Inventory” could include no more than one “Part.”

   d. A and B only.
   e. B and C only.
   f. A, B, and C.

29. Once the first draft of the class diagram is completed, the team undertakes an internal review. Which of the following is a valid criticism and rationale?
   a. Classes “Parking” and “Car_maintenance” should be combined into a single class, since they have the same attributes and inherit from the same superclass.
   b. Class “Hash table” introduces too much detail into the model, since that is an implementation detail that should not be decided in conceptual design.
   c. Method “price” does not need to be defined in class “Refuel” since a method with the same name and interface is inherited from superclass “Services.”
   d. A and B.
   e. B and C.
   f. A, B, and C.


Registered Customer
~~C~~ashier_terminal
Gas Station
Bill
pay _monthlybycash(account number, amount)
Credit_bill(account number, amount)
IsClientRegistered(account number)
get _bill( )
update_bill(amount, payment date)
display_message(text)
[information is ok]

The figure above shows the first draft of a sequence diagram for the gas station system. Use it to answer questions 7 and 8.

30. Which of the following statements are valid interpretations of the sequence diagram?
   a. An object of type “Gas Station” attempted to send an “IsClientRegistered” method to a “Bill” object, but was unsuccessful.
   b. The update_bill method is sent only if the information is OK.

c. An object of type “Bill” is created at the time a “Gas Station” class sends the update_bill method.
d. A and B.
e. B and C.
f. A, B, and C.

31. TRUE or FALSE: A “Cashier_terminal” object will send a “credit_bill” method only after having received a pay_monthlybycash method.

Once the conceptual design is finished, the team undertakes an internal review before sending it to the gas station owner.

32. The first draft of the conceptual design has the following characteristics. Which are NOT appropriate?
    a. It is written in a formal design notation, so that functionality can be specified as precisely as possible.
    b. It makes multiple references to the requirements document, in order to provide the reason for including certain system components.
    c. It uses some of the built-in features of C++, in order to better make the argument for choosing this language for use in the implementation.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

33. One problem found during design review was that functionality was omitted from the design, even though it was included in the requirements. To avoid this problem in the future, the development team should consider:
    a. Choosing an improved notation for the design.
    b. Adding configuration management to validate the requirements.
    c. Investing more time in understanding the user's requirements.

Based on the conceptual design, the gas station owner responds with some critiques of the user interface.

34. The main critique is that it will be hard for the cashiers to learn the system because each screen is laid out differently. For example, on the cashier's initial screen the options are laid out across the bottom. But when the cashier is inputting data about the purchase of car parts, the cashier's options are on the left side of the screen from top to bottom, which is confusing. What the owner is really saying is that the system needs a consistent:
    a. Metaphor
    b. Mental model
    c. Navigation rule
    d. Look
    e. Feel

The development team has decided to divide the system into three subsystems:

- A gas purchase subsystem, that takes care of customer interaction with the gas pumps;
- A cashier subsystem, that interacts with the cashier to accept payment for the purchase of car parts and gasoline;
- A tracking subsystem, that logs all purchases and tracks the inventory remaining.

Each of these subsystems is expected to be a relatively complicated system in its own right, but this design was chosen to minimize the communication among subsystems. Both the gas purchase and cashier subsystems will communicate with the tracking subsystem, but not with each other.

35. Which of the following best describes the design at this level of abstraction?
    a. High coupling, high cohesion
    b. High coupling, low cohesion
    c. Low coupling, high cohesion
    d. Low coupling, low cohesion

36. TRUE or FALSE: The above combination of coupling and cohesion will make programming and maintenance easier than if the system had been designed otherwise.

37. In the initial design, the gas purchase subsystem is assumed to handle all of the details of the purchase, and need send only a record containing the amount of gas purchased, purchase price, and time of purchase to the tracking subsystem. The relationship between the gas purchase subsystem and the tracking subsystem is best described as:
    a. Content coupled
    b. Control coupled
    c. Stamp coupled
    d. Logically cohesive
    e. Temporally cohesive
    f. Functionally cohesive

38. The team leader realizes the team does not have much experience building systems able to handle concurrency, as the gas purchase subsystem will have to do. He therefore decides that the best way to proceed will be for the team to develop a basic design for the subsystem that demonstrates that concurrency can be handled, but does not include the full range of customer functionality. The full range of functionality can be added once the team is sure they have achieved an adequate design for handling concurrency. This strategy is an example of:
    a. A prototype design
    b. A throwaway prototype
    c. Fault-tree analysis
    d. Design by contract

39. The gas purchase subsystem needs to be able to handle the situation in which the customer pays by credit card, but the remote system that validates the credit card information is unreachable, perhaps because of a temporary network failure. This situation is an example of:
    a. An exception
    b. A fault
    c. A failure

40. It would be reasonable to design the system so that, the first time the situation described in question 39 occurs, the system responds by:
    a. Retrying
    b. Correcting
    c. Reporting
    d. Active fault detection
    e. Passive fault detection

41. TRUE or FALSE: The team lead has monitored several design metrics over the course of the design effort. As the design seems to be nearing completion, he reviews his notes and notices that the metric for “weighted methods per class” for class “Parking” has increased from 6 to 12, at the start of the detailed design phase. The next step should be to split “Parking” into several classes, each with fewer methods.

42. After more work has been done on the design, the team leader decides it is time to hold a program design review. Which of the following need NOT be invited?
    a. The requirements analysts
    b. The developers
    c. The gas station owner
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

43. Which of the following items of information are NOT appropriate for inclusion in the final design?
    a. The maximum cost the gas station owner is willing to pay for the system
    b. The layout of the cashier's screen
    c. The general layout of the network that supports communication within and among subsystems
    d. None of the above (all are appropriate)
    e. a and b
    f. a and c
    g. b and c
    h. a, b, and c

44. Which of the following items of information are NOT appropriate for inclusion in the final design?
    a. The maximum amount of time the cashier can be made to wait for a response from the system.
    b. How the record of each day's transactions will be archived.
    c. What happens if the network connections to any of the gas pumps are severed.
    d. None of the above (all are appropriate)
    e. a and b
    f. a and c
    g. b and c
    h. a, b, and c

Review Exam 2 Answers

1. e; Choice B is false because the requirements cover only what functionality is implemented, not how. [Section 4.1]
2. d; Since both the developers and owner may revise their concepts of the system as they learn more about it, the requirements should not be assumed to describe the final system exactly. However, the requirements provide a useful starting point for discussing these revisions over the course of the implementation. [Section 4.1]
3. a; Event tables are not well suited to concurrent environments; Z is very formal and does not assist communication with people who are unfamiliar with the notation; OO specifications do not contain any specific methods for ensuring completeness. [Sections 4.4-4.5]
4. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
5. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
6. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
7. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
8. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
9. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
10. TRUE; The requirements specifications should contain anything relevant to how the system will interact with its environment. [Section 4.2]
11. e; Choice B contains details about how the system should be implemented [i.e. using a linked list] which is outside the scope of the requirements. [Section 4.1]
12. a; Choice B is a functional requirement; Choice C describes something outside the control of the system. [Section 4.1]
13. b and d; The phrase “relevant information” is ambiguous. (How is the developer to know what information is relevant?) This ambiguity also serves to make the requirement unverifiable. [Section 4.3]
14. d; The phrase “easy to add new functionality” is unverifiable. How is “easy” defined? What types of functionality? [Section 4.3]
15. e; The formula described can be verified for correctness, and is not ambiguous. [Section 4.3]
16. d; The phrase “easy to use” is unverifiable. The requirements should be rewritten using a measurable criterion, e.g. that a new user must have less than a certain number of faults, or that a new user should not take more than a specified amount of time to complete the transaction. [Section 4.3]
17. c; The requirements needs to identify factors affecting availability. For example, presumably the system does not work if there is a power failure. But how about more mundane matters, such as routine maintenance? For example, if receipts are printed, the paper spool presumably has to be replaced some time. [Section 4.3]
18. c; Where possible, requirement measures should be categorized by requirement type, so that it can be understood whether change and uncertainty are product-wide or rest solely with a specific type of requirement. [Section 4.10]
19. g; The conceptual design addresses issues such as what the system looks like to users, where the data comes from, and what happens to the data in the system. [Section 5.1]

20. g; The technical design is better suited to describing issues such as major hardware components and data structures. [Section 5.1]
21. c; Choice C is not valid since Object-Oriented design is not particularly well-suited to describing data flow (although data-oriented decomposition is). [Section 5.2]
22. f; Pipe and filter is not an appropriate choice because it is not well suited to interactive applications. Implicit invocation is not a good choice because one of its drawbacks is that there is no assurance that a component will respond to an event. [Section 5.3]
23. FALSE; Section 5.3 describes how this process is likely to be iterative.
24. E. No ordering is implied by the high-level use case diagram. [Section 6.3]
25. C. The triangle implies specialization. That is, “Parking” is a specific type of “Billing Services.” [Section 6.3]
26. D. Combining “Cashier” and “Customer” for the reason given in choice C is not valid, since these actors can have unique roles in the scenarios of which they are both part. [Section 6.3]
27. E. The cardinality notation for these two classes signifies that an instance of class “Message” can be associated with 0 or more “Registered customer” objects, and an instance of class “Registered customer” can be associated with 1 or more “Message” objects. [Section 6.5]
28. F. All of the choices are consistent with the class diagram. (A subclass inherits its parent’s associations.) [Section 6.5]
29. B. Methods and attributes with the same name in different classes can still have different definitions and values, common to all instances of the relevant class. [Section 6.5]
30. E. An object can call one of its own methods, as is indicated by the notation of a method arrow beginning and ending with the same object. [Section 6.5]
31. TRUE. Sequence diagrams convey chronological information, with methods lower in the diagram occurring after those closer to the top. [Section 6.5]
32. e; Conceptual design should be written in the customer's language and be independent of the implementation. [Section 5.1]
33. b; Configuration management is concerned with demonstrating that documents at each stage are compatible with documents from other stages; if there were a closer correspondence between requirements and design less functionality may have been lost. [Section 4.1]
34. d; The “look” of a system refers to “characteristics of the system's appearance that convey information to the user.” [Section 5.4]
35. c; The components are loosely coupled since they are relatively independent, with some interconnections. The components are cohesive since all of their subcomponents will be directed toward (and presumably essential for) supporting a particular functionality of the GSCS. [Section 5.5]
36. TRUE; Components are easier to understand if they are not intrinsically tied to others (i.e. not tightly coupled). Similarly, cohesive components, with logically related subcomponents, are generally easier to understand than non-cohesive ones. [Section 5.5]
37. c; Components exhibit stamp coupling when a data structure is used to pass information from one component to another. [Section 5.5]
38. a; Prototypes omit some details of functionality and performance, so that particular system aspects can be focused on. The omitted details are then filled in later (unlike throwaway prototyping, in which the final system is not built directly from the initial prototype). [Section 5.6]

39. a; The situation is not a fault or failure because it does not represent a defect in the GSCS. It is an exception because it does not occur in normal system operation. [Section 5.5]
40. a; Since the network problems may only be temporary, it makes sense to restore the system to its previous state and try contacting the credit card system again, before taking more extreme measures. [Section 5.5]
41. FALSE. The fact that the metric has increased is not enough information to justify automatically splitting the class. It should be monitored closely, and compared to other classes in the system and other classes this team may have had experience with in the past. If the value is high relative to other classes, it is a strong indication that the class may be more difficult to implement than others. [Section 6.7]
42. c; The program design review allows designers to receive feedback from other designers, analysts, and programmers. The customer of the system does not have a role to play. [Section 5.7]
43. a; The design should describe the system in such a way that it can be validated whether the system will meet the requirements of the user. The design should address how users interact with the system (including display-screen formats) and network issues (such as topology). [Section 5.8]
44. d; The design should describe the system in such a way that it can be validated whether the system will meet the requirements of the user. The design should address how users interact with the system (including performance constraints, and how output are stored) and network issues (including prescriptions for system integrity in the event of a network failure). [Section 5.8]

Chapter 7: Writing the Programs

Learning Objectives:
After studying this chapter, you should be able to:

- Describe why programming standards and procedures are important for you and for others.
- Define the two types of reuse, producer and consumer.
- Understand the characteristics that influence whether or not a component can be reused.
- Understand how the design is used to frame the code.
- Understand what should be included as part of the internal and external documentation.

Summary:
This chapter addresses issues in implementing the design to produce high-quality code. Standards and procedures are discussed and some simple programming guidelines are suggested. Examples are provided in a variety of languages, including both object-oriented and procedural. The chapter contains discussions of the need for program documentation and an error-handling strategy. This chapter does not teach how to program; rather, it explains some of the software engineering practices that should be kept in mind as code is written.

The task of writing the programs that implement the design can be daunting for several reasons. First, the designers may not have addressed all of the idiosyncrasies of the platform and programming environment; structures and relationships that are easy to describe with charts and tables are not always straightforward to write as code. Second, code must be written in a way that is understandable not only to the author when it is revisited for testing but also to others as the system evolves over time. Third, programmers must take advantage of the characteristics of the design's organization, the data's structure, and the programming language's constructs while still creating code that is easily reusable.

When writing code, the following items should be considered:

- organizational standards and guidelines
- reuse of code from other projects
- writing code to make it reusable on future projects using the low-level design as an initial framework, and moving in several iterations from design to code
- incorporating a system-wide error-handling strategy
- using documentation within programs and in external documents to explain the code's organization, data, control and function, as well as design decisions
- preserving the quality design attributes in the code
- using design aspects to suggest an implementation language.

Many corporate or organizational standards and procedures focus on the descriptions accompanying a collection of programs. Program documentation is the set of written descriptions that explain to a reader what the programs do and how they do it. Internal documentation is descriptive material written directly within the code. All other documentation is external documentation. Internal documentation includes summary information to describe its data structures, algorithms and control flow. With external documentation, the summary information is provided from a system rather than component perspective.

Exercises:

1. A *stack* is a data structure used to store elements. A stack is a last-in, first-out data structure. That is, the last element placed on the stack is the first element that can be removed from the stack. Elements can be placed on or removed from the top of the stack only. The allowable operations for a stack are *empty*, *full*, *push*, *pop* and *top*.

   The empty operation returns true if there are no elements in the stack, false otherwise.

   The full operation returns true if the stack is filled to capacity, false otherwise.

   The push operation takes an element as an argument and places the element on top of the stack, if the stack is not full. If the stack is full, the push operation returns an error.

   The pop operation removes an element from the top of the stack, if the stack is not empty. If the stack is empty, the pop operation returns an error, otherwise, the top element is returned.

   The top operation returns the element on the top of the stack without removing the element from the stack, if the stack is not empty. The top operation returns an error if the stack is empty.

   Use an array to implement a stack data structure whose elements are integers. The stack may contain a maximum of 100 elements. Keep in mind the guidelines for programming style that were presented in this chapter.

2. One difficulty with reuse is selecting an appropriate component. Describe a strategy for finding a reusable component. What guidelines or styles would help in the process of selecting a component for reuse?

3. Explain the relationship between the design and implementation. Why is it important to match the implementation to the design? What would you do to keep the two consistent?

4. Consider a case where you have attempted to reuse code written by someone else. What kind of reuse was it? What problems did you encounter? How did you resolve the problems? Are there any guidelines in this chapter that may have helped to eliminate or mitigate the problems you encountered?

5. When writing code, many people are usually involved. Writing code usually requires a great deal of cooperation and coordination. It is important for others to be able to understand what you have written, why you have written it, and how your work fits in with their work. For these reasons, many organizations have coding standards and procedures. Using the guidelines from this chapter, write a set of coding standards for a language of your choice. Explain why you have included the standards you have chosen.

Answer Guidelines:

1. To implement the stack data structure, you should have followed the guidelines presented in the chapter. You should use meaningful variable names, provide good documentation of your code, use efficient algorithms, and maintain good design principles (low coupling and high cohesion).
2. When reusing a component, you may want to examine the documentation, look at the test history, or test the software before you actually commit to using it. It is important to understand whether or not you will have access to the source code, to know who is responsible for changes and to understand the limitations of the reusable component. In Section 7.3, some of the key characteristics you should consider when selecting a reusable component are described. Use this list of characteristics to develop your strategy for selecting a component. How would the strategy be different for white-box versus black-box reuse?
3. The code should implement the design. Design characteristics such as high cohesion, low coupling and well-defined interfaces should be program characteristics as well. It is important that the design and code match for other activities such as maintenance and testing. To maintain traceability, you may want to include design information in the program comments. Configuration management may also help to maintain consistency between the code and design. Section 7.1 describes the relationship between design and implementation.
4. There are many types of problems that may be encountered when you are a consumer of a reusable component. The documentation may be misleading or incorrect. There may be missing functionality that is required by your system. You must determine how to fit a reusable component into the design of the new system. These are only a few of the problems that may occur. Section 7.3 describes characteristics that should be considered when reusing components. Based on the problems that you encountered and the guidelines and programming styles described in this chapter, can you describe ways that the reusable component could have been changed to eliminate or mitigate the problems that you encountered?
5. To write the standards, make the guidelines presented in the chapter operational. For example, to make the guideline of meaningful variable names operational, you might have a standard which requires all variable names to be greater than 5 characters and less than 10 characters. Your reasons for including this standard might be that variable names less than 5 characters are cryptic and anything over 10 characters may be difficult to remember.

Chapter 8: Testing the Programs

Learning Objectives:
After studying this chapter, you should be able to:

- Define different types of faults and how to classify them.
- Define the purpose of testing.
- Describe unit testing and integration testing and understand the differences between them.
- Describe several different testing strategies and understand their differences.
- Describe the purpose of test planning.
- Apply several techniques for determining when to stop testing.

Summary:
This chapter explores several aspects of testing programs. A distinction is made between conventional testing approaches and the cleanroom method. A variety of testing strategies are presented. The chapter also presents definitions and categories of software problems and discusses how orthogonal defect classification can make data collection and analysis more effective. The difference between unit testing and integration testing is explained. The chapter also describes the need for a testing life-cycle and describes how automated test tools and techniques can be integrated into it.

Testing is not the first place where fault-finding occurs; requirements and design reviews help to ferret out problems early in development. But testing is focused on finding faults, and there are many ways to make testing efforts more efficient and effective. It is important to understand the difference between a fault (a problem in the requirements, design, code, documentation or test cases) and a failure (a problem in the functioning of the system). Testing looks for faults, sometimes by forcing code to fail and then seeking the root cause. Unit testing is the development activity that exercises each component separately; integration testing puts components together in an organized way to help isolate faults as the combined components are tested together.

Testing is both an individual and a group activity. Once a component is written, it can be inspected by some or all of the development team to look for faults that were not apparent to the person who wrote it. The research literature clearly shows that inspections are very effective at finding faults early in the development process. But it is equally clear that other techniques find faults that inspections often miss. So it is important for team members to work with the team in an egoless way, using the many available methods, to find faults as early as possible during development.

The goal of testing is to find faults, not to prove correctness. Indeed, the absence of faults does not guarantee correctness. There are many manual and automated techniques to help find faults in code, as well as testing tools to show how much has been tested and when to stop testing.

Exercises:

1. Examine faults from code that you have written. For each fault, identify the type of fault (as in Section 8.1, Types of Faults) and classify the fault using a defect classification. Provide the details of the defect classification used. (You may use the IBM defect classification presented

in Table 8.1 or the one from HP illustrated in Figure 8.1.) Describe any difficulties encountered in classifying the faults.

2. Based on the faults identified in the previous question, which type of fault occurred most frequently? How might you change your software development approach to eliminate or reduce the occurrence of this type of fault?
3. Describe the differences between unit and integration testing. Give the goals for each type of testing and describe when and how each should occur.
4. Describe the differences between object-oriented and traditional testing.
5. Choose a piece of code and write test cases for the code to satisfy the requirements of statement testing. Write the test cases for all-uses testing. Which testing strategy is stronger? Which strategy requires more test cases?

Answer Guidelines:

1. Chapter 8 describes many different types of faults. The purpose of this exercise is to give you a better understanding of how these descriptions can be used to identify and understand faults in actual code. It will also help you to understand the difficulties in classifying defects. Sometimes it is difficult to classify faults, especially when the defect classification is not orthogonal. The chapter presents two defect classifications. You may use one of these classifications or any other, reasonable classification. Be sure your answer clearly describes the defect classification you are using. When answering this question first decide which type of fault you have found. Then, determine if the fault is one of omission or commission. Finally, based on the fault type and your classification of omission or commission, use your defect classification to classify the fault. Did you have difficulties in determining the fault type? Did any of your faults seem to fit in multiple categories?
2. Answers to this question will vary depending on your fault profile. Your fault profile can be used to identify areas of improvement for yourself. Based on your profile, which type of fault had the highest frequency? Would any of the techniques described in this chapter or previous chapters be useful in helping you to reduce the frequency of this type of fault? Which type of fault occurs least frequently? Have you done anything in the past to prevent this type of fault from occurring?
3. The main purpose of unit testing is to make sure that the component is functioning properly. The component is tested in isolation to make sure that the inputs produce the expected outputs. The main purpose of integration testing is to verify that the system components work together as specified by the design. Integration testing occurs after unit testing. Sections 8.3 and 8.4 describe unit and integration testing in greater detail.
4. Section 8.5 addresses the difference between testing object-oriented systems and traditional systems. Most of the techniques used for traditional testing also apply to object-oriented systems. Object-oriented programs have special characteristics that need several additional steps. Some of the characteristics that must be considered with OO programs that may not be included with traditional testing techniques are: missing objects, unnecessary classes, missing or unnecessary associations, or incorrect placement of associations or attributes. Test case adequacy must also be considered more carefully with OO systems. As Perry and Kaiser (1990) found, when a subclass is added or modified, the inherited methods from the ancestor superclasses must be retested. As noted by Graham (1996a), objects tend to be small

and low in complexity. However, the complexity often is pushed to the interfaces among components. This shift of complexity means that unit testing may be easier with OO systems, but integration testing must be more extensive.

5. Section 8.3 describes the different types of test strategies for test thoroughness. Statement testing and all-uses testing are two of the options described. With statement testing, every statement is executed at least once in a test case. With all-uses testing, the test set includes at least one path from every definition to every use that can be reached by the definition. In general, all-uses is stronger and requires more test cases than statement testing.

Chapter 9: Testing the System

Learning Objectives:
After studying this chapter, you should be able to:

- Describe how system testing differs from unit and integration testing.
- Classify tests as function testing, performance testing, acceptance testing or installation testing.
- Understand the purposes and roles of function testing, performance testing, acceptance testing, and installation testing.
- Define software reliability, maintainability and availability.
- Describe different techniques for measuring reliability, maintainability and availability.
- List the different types of test documentation and know what items belong in test documentation.
- Understand the special problems associated with testing safety-critical systems.
- Describe the principles of Cleanroom and how it differs from conventional testing.

Summary:
This chapter looks at the system testing process: its purpose, steps, participants, techniques and tools. The chapter describes the principles of system testing, including reuse of test suites and data, and the need for careful configuration management. The concepts introduced include function testing, performance testing, acceptance testing and installation testing. The chapter examines the special needs of testing object-oriented systems. Several test tools are described, and the roles of test team members are discussed. The reader is introduced to software reliability modeling. The issues of reliability, maintainability and availability are discussed. The chapter describes how to use the results of testing to estimate the likely characteristics of the delivered product. Several types of test documentation are described.

Testing the system is very different from unit and integration testing. When unit testing components, the developer has complete control over the testing process. The developer creates the test data, designs the test cases, and runs the tests. When integrating components, the developer sometimes works individually, but often collaborates with a small part of the test or development team. However, when testing a system, the developer works with the entire development team, coordinated and directed by the test team leader.

The objective of unit and integration testing is to ensure that the code implements the design properly. In system testing, however, the objective is to ensure that the system does what the customer wants it to do. Test procedures should be thorough enough to exercise system functions to everyone's satisfaction: the user, customer, and developer.

The steps involved in system testing include function testing, performance testing, acceptance testing, and installation testing. Each step has a different focus. Function testing checks that the integrated system performs its functions as specified in the requirements. Performance testing compares the integrated components with the nonfunctional system requirements. Acceptance testing assures the customers that the system they requested is the

system that was built for them. Installation testing allows users to exercise system functions and document additional problems that result in the actual operating environment.

Often, a system is tested in stages or pieces. System testing must also take into account the several different system configurations that are being developed. A system configuration is a collection of system components delivered to a particular customer. During testing, configuration management, the control of system differences to minimize risk and error, is especially important. Configuration management helps to coordinate efforts among the testers and developers.

Techniques such as Cleanroom require a great deal of team planning and coordination, in developing the box structures and in designing and running the statistical tests. And the activities involved in acceptance testing require close collaboration with customers and users; as they run tests and find problems, the team must quickly determine the cause so that corrections can allow testing to proceed. Thus, whereas some parts of development are solitary, individual tasks, testing the system is a collaborative, group task.

Exercises:

1. How does system testing differ from unit and integration testing?
2. Explain the purposes and roles of function testing, performance testing, acceptance testing, and installation testing.
3. What is the difference between verification and validation? Which types of testing address verification? Which types of testing address validation?
4. Describe the principles of Cleanroom and how it differs from conventional testing.
5. Read the press release and failure report for the Ariane-5 Flight 501. An electronic copy of the failure report is available at http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html. The joint ESA/CNES press release is available at http://www.esrin.esa.it/htdocs/tidc/Pres/Press96/pres19.html. What kinds of tests might have exposed the problems that caused each of the failures?

Answer Guidelines:

1. The emphasis for unit and integration testing is to make sure the code implements the design properly. With system testing, the focus is shifted to the customer. System testing looks to verify that the system implements the requirements properly.
   For more details on the differences, re-read Section 9.1.
2. The purpose and roles of the different types of testing are presented throughout the chapter. For each type of system testing mentioned, describe the purpose and role. Explain when in the system testing process each type of test should occur. Describe how and why each type of test is performed.
3. A verified system implies that the system operates the way the designers intended it to operate. A validated system implies that the system meets the customers' expectations. The various types of unit and integration tests focus on verification. System testing focuses on validation. Review the descriptions of the types of unit and integration tests and the types of system testing to determine which tests contribute to verification and validation.
   Section 9.1 contains more information on verification and validation.
4. Cleanroom reflects the ideas used in the manufacturing of chips. The goal is to keep faults at a minimum. For software, the goals are to certify software before unit testing and to produce as few faults as possible.

With Cleanroom, verification replaces unit testing. Cleanroom also makes use of statistical testing.
Section 9.9 describes the principles and the advantages and drawbacks to the Cleanroom process in greater detail. Use this information to compare the Cleanroom process against traditional testing.

5. The most obvious failure from the Ariane-5 flight was the explosion of the space rocket itself. The failure report describes additional faults and failures that contributed to the explosion. Use the descriptions of the types of testing in this chapter to determine which tests may have uncovered the faults. Explain how the testing would have uncovered the fault. Be sure to consider whether or not the type of test you describe would have been feasible.

Chapter 10: Delivering the System

Learning Objectives:
After studying this chapter, you should be able to:

- Describe different types of training and training aids.
- Understand the differences between user and operator training.
- Describe special training needs and guidelines for training.
- Describe the types of documentation needed for training.

Summary:
This chapter discusses the need for training and documentation, two issues key to successfully transferring the system from the developer to the user. The chapter presents several examples of training and documentation that could accompany a software system.

Many software engineers assume that system delivery is a formality. However, even with turnkey systems (where the developers hand over the system to the customer and are not responsible for its maintenance), delivery involves more than putting the system in place. It is the time during development when the development team helps users to understand and feel comfortable with the product. If delivery is not successful, users will not use the system properly and may be unhappy with its performance. In either case, users are not as productive or effective as they could be, and the care taken to build a high-quality system will have been wasted.

As the system is designed, aids that help users learn to use the system are planned and developed. Accompanying the system is documentation to which users refer for problem-solving or further information. Training and documentations should be done from two perspectives: the user and the operator. Sometimes, the same person is both user and operator. However, user and operator tasks have very different goals, so the training for each job should emphasize different aspects of the system.

Training can be done in many ways. No matter how training is provided, it must offer information to users and operators at all times, not just when the system is delivered. At some time, if users forget how to access a file or use a new function, training includes methods to find and learn this information. Formal documentation, icons and on-line help, demonstrations and classes, and expert users are examples of training aids that may be provided.

Exercises:

1. Examine the documentation and training resources for a software system of your choice. What resources are available to the users? What resources are available to the operators?
2. What kinds of training aids and resources would be useful for users with little or no computer experience? Give examples of aids or documentation meant for novice users.
3. Suppose a system has functions that are rarely executed by users or operators. What types of resources would be appropriate for these rarely used functions?
4. Often, the training and documentation needs for novice users are very different from the needs of expert users. Give examples of cases where training aids or documentation for one user group are inappropriate for the other.

5. Think about experiences you have had with training and documentation resources. Are there any cases where these aids have interfered with your usage of the software system? If so, how would you change the documentation or training aid?

Answer Guidelines:

1. This exercise is intended to help you to identify the different training and documentation resources available with software systems. Use the descriptions of training and documentation presented in this chapter to help you identify the resources available with the system of your choice. You should be able to distinguish between resources that are meant for users and those resources that are meant for operators.
2. In this exercise, you should focus on the training and documentation aids appropriate for novice users. Throughout the chapter, there are descriptions of various resources. When reviewing these descriptions, think about how each resource described might be useful for an inexperienced user. Try to find examples of support for novice users in software systems that you have used.
3. The knowledge gained in training can be forgotten easily over time if the system functions are not exercised regularly. There are several training options that may be useful in this situation. You should review the different resources described throughout the chapter and comment on the resources that would be useful in this situation. In addition, you may want to include examples from your own experiences.
4. To answer this question, review the training aids and documentation examples presented throughout the chapter. As you review the resources, think of examples that you have experienced in software that you have used. Categorize these examples as resources for novices or resources for expert users.
   Use this information to describe reasons why resources for one group may not be appropriate for the other user group. For example, “wizards” may be useful for novices; however expert users may find them cumbersome to use. Similarly, documentation meant for expert users may be incomprehensible to novice users.
5. The answers to this question will vary based upon your experiences with training aids and resources. Use the chapter descriptions to help you identify different types of training aids. Have you encountered resources that have hindered you use of a software system? For example, was the user documentation incorrect? Were there aids that were meant for novice users that you were unable to disable? Was information missing from the documentation that made the system impossible to use? Once you have answered these questions, think about things that you would do to eliminate some of the problems that you have encountered.

Review Exam 3

The following questions are in reference to a hypothetical "Gas Station Control System" (or GSCS) that will be used to help manage an American-style gasoline or service station. Our hypothetical gas station basically provides two services:

- There is a small store that carries car parts. Inside the store is at least one cash register, operated by a cashier who is an employee of the gas station.
- There are a number of gas pumps, at which customers can park their cars, interact with the system to pay via credit card, and then pump their own gas. Alternatively, the customer can pay for his or her gas via cash or credit card by going into the store and paying directly to the cashier.

Thus the GSCS has two main classes of users. The first is the cashier, who uses the GSCS to record purchases of car parts by customers. The GSCS must allow the cashier to enter the type and number of parts purchased, then compute the total purchase price and handle the payment. Customers purchasing gasoline are the second type of user. These customers interface with the system at the gas pump, by specifying the amount and type of gas they will buy, paying either at the pump or to the cashier, and then pumping the gas themselves.

The system also has to interact with other automated systems to perform its tasks. For example, in order to accept credit card payments, the GSCS must interface with a system maintained by the credit card company. The credit card system is responsible for checking that the customer's account is in good standing and can accommodate the amount of the purchase, and for debiting the customer's account and eventually reimbursing the gas station. The operation of these external systems is beyond the scope of the GSCS, although the GSCS needs to know how the external systems will communicate the success or failure of their tasks.

The team has finished the design of the system and has begun coding.

1. Danielle, the development team leader for the GSCS, has decided to emphasize the use of corporate software guidelines. Which of the following statements best describe the benefits the team might expect from documenting the code and making it readable?
   a. The documentation improves the efficiency of the code.
   b. The documentation provides traceability to design components.
   c. The documentation improves the organization of the code.
   d. a, b, and c
   e. a and b only
   f. b and c only

2. Which of the following statements does NOT describe a reasonable rationale for passing a variable by reference to a function?
   a. The changes to the variable values are needed after the function terminates and the variable size is small.
   b. The changes to the variable values are not needed after the function terminates and the variable size is small.
   c. The changes to the variable values are needed after the function terminates and the variable size is large.
   d. The changes to the variable values are not needed after the function terminates and the variable size is large.

3. A function with an input domain of the set of all real numbers is tested with sets of positive integers, negative integers and 0. In all of the tests, the function performs properly. It is safe to assume that the function has been tested thoroughly and will perform properly. (TRUE/FALSE)

In the following program fragments from the GSCS, identify violations (if any) of good programming style. Use the following choices in your response:
(a) Generality
(b) Efficiency
(c) Formatting
(d) Documentation
(e) No violations

4.
```
void PrintPartFile(){
/* Open the parts.dat file.  Print each line to standard output.  */
/* Close the file. */
ifstream PartFile ("parts.dat");
char line[100];

while (PartFile.getline(line,100))
cout << line << "\n";

PartFile.close();
}
```

5.
```
int ValidateParts (PartList &parts, PartList &master){
/* Validate and count the parts in the part list (parts).                */
/* If a part in the list (parts) does not exist in the master list,      */
/*    return -1.                                               */
/* If all parts in the list (parts) exist in the master list,            */
/*    return the sum of the quantities of each part in the list (parts). */

int total = 0;

for (int i=0; i < parts.getcount(); i++)
if (!master.Exists(parts[i])) return -1;
for (int j=0; j < parts.getcount(); j++)
total += parts[i].getquantity();
return total;
}
```

6.
```
const int MAXSALES=100000; /* maximum number of sales stored */
const int MAXNAME=100; /* maximum name size */

struct CashierRecord{
char name[MAXNAME]; /* cashier's name */
float sales[MAXSALES]; /* sales made by the cashier */
int count; /* number of sales stored in the sales array */
} Cashier;

Cashier cashier1;

/* SumSales sums the sales transactions for the cashier. */
/* The return value is the sum of all transactions. */
```

```
float SumSales(Cashier c){
float total; /* variable that will store return value */
int i; /* loop counter */

total = 0;
/* sum the sales transactions */
for (i=0; i < c.count; i++)
total += c.sales[i];
return total;
}
.
.
.
int main(){
Cashier cashier1;
.
.
.
SumSales(cashier1);
.
.
.
}
```

7.
```
void Print(istream &is, ostream &os){
/* Print each line of the input stream (is) to the output stream (os).  */
char line[MAXLINE];

while (is.getline(line,MAXLINE))
os << line << "\n";
}
```

8.
```
/* Print each value of the values array on a separate line. */
/* Print the total on a separate line after the values. */
for (int i=0; i<count; i++)
total += values[i];
cout << values[i] << "\n";
cout << "Total = " << total << "\n";
```

During code reviews of the GSCS, the following faults were identified. Classify the type of fault in each code fragment.

9.
```
int n;
float x[1000];

x[i] = (1 - 2/(n-1)) * x[i - 1] + 2/(n-1) * x[i];
```

In this fragment, since n is an integer, the division of 2/(n-1) returns an integer value.

a. boundary fault
b. initialization fault
c. computation/precision fault
d. b and c only
e. a and b only
f. none of the above

10. List::~List(){
/* delete all of the elements in the list */
for (int i=1; i < count; i++)
delete list[i];
}

In this fragment, the first item of the list (list[0]) is not deleted.

a. initialization fault
b. documentation fault
c. precision fault
d. b and c only
e. a and b only
f. none of the above

11. int list[10];
for (int i=0; i<=10; i++) list[i] = i;

In this fragment, the loop includes an operation on list[10] which is not part of the array.

a. initialization fault
b. precision fault
c. capacity or overload fault
d. a, b, and c
e. none of the above

12. float list[100];
float xrange=list[99]-list[0];

for (int i=0; i<100; i++)
list[i] = (list[i]-list[0])/xrange;

In this fragment, list[0] is changed after the first iteration. All items of the array will be 0 after the array is executed. Also, there is no check on xrange. It may evaluate to 0 causing a division by zero.

a. computation/precision fault
b. initialization fault
c. capacity or overload fault
d. a, b, and c
e. none of the above

13. Given the following assertions:

A_1: (*T* is an array) & (*T* is of size N) & (*S* is an array) & (*S* is of size N)

A_{end}: (*T*' is an array) & (*T*' is of size N) & ($\forall$i, 0 $\le$ i $\le$ N, T'(i) = S(i))

Choose the statement that best describes what is happening between the two assertions.

a. The values of array S are being assigned to the array T'.
b. The values of array T are being assigned to the array S.
c. The values of array S are being added to the values of array T'.
d. The values of array T are being added to the values of array S.
e. None of the above.

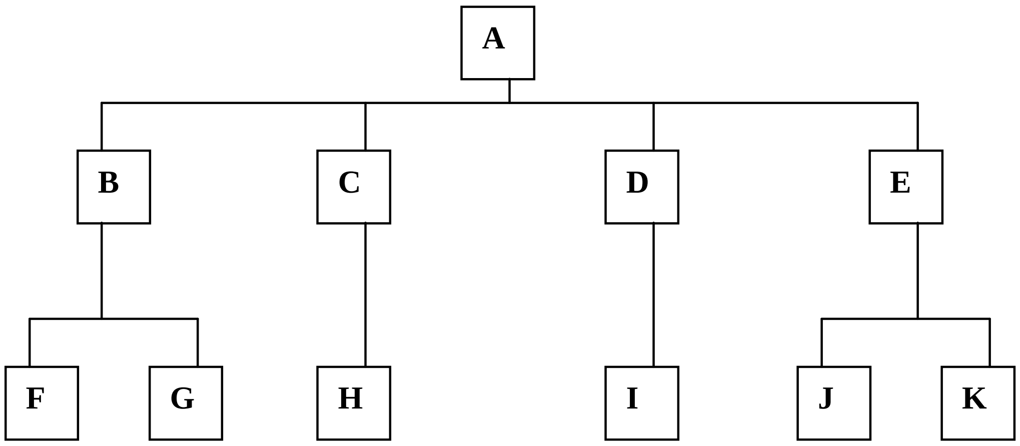
14. Suppose the main objective of the GSCS development is to get a working system to show the customer as soon as possible. The best testing approach to choose would be:

a. Bottom-up testing
b. Top-down testing
c. Big-bang testing

d. a or b
e. a or c

Implementation of the GSCS is complete and the development has entered the testing phase.

The figure below shows the component hierarchy of the GSCS system. Use this figure to identify the testing strategy indicated by the sequences given. The “;” is used between test sets and each test set is represented as a comma-separated list. For example, the sequence {F,G};{B,F,G} means that components F and G were tested first. Then, components B, F and G were tested.



15.{J};{K};{I};{H};{G};{F};{B};{C};{D};{E};{A};{A,B,C,D,E,F,G,H,I,J,K}
   a. Top-down testing
   b. Bottom-up testing
   c. Sandwich testing
   d. Big-bang testing
   e. Modified top-down testing

16.{A};{A,B,C,D,E};{A,B,C,D,E,F,G,H,I,J,K}
   a. Top-down testing
   b. Bottom-up testing
   c. Sandwich testing
   d. Big-bang testing
   e. Modified top-down testing

17.{F};{G};{H};{I};{J};{K};{B,F,G};{C,H};{D,I};{E,J,K};{A,B,C,D,E,F,G,H,I,J,K
}
   a. Top-down testing
   b. Bottom-up testing
   c. Sandwich testing
   d. Big-bang testing
   e. Modified top-down testing

18.{A};{B};{C};{D};{E};{A,B,C,D,E};{F};{G};{H};{I};{J};{K};{A,B,C,D,E,F,G,H,I
,J,K}
   a. Top-down testing
   b. Bottom-up testing

   c. Sandwich testing
   d. Big-bang testing
   e. Modified top-down testing

19. {A};{F};{G};{H};{I};{J};{K};{B,F,G};{C,H};{D,I};{E,J,K};{A,B,C,D,E,F,G,H,I,J,K}
   a. Top-down testing
   b. Bottom-up testing
   c. Sandwich testing
   d. Big-bang testing
   e. Modified top-down testing

20. In a function of a component in the cashier subsystem of the GSCS, a variable does not get initialized properly. Which type of testing would be most likely expose this defect?
   a. unit testing
   b. integration testing
   c. acceptance testing
   d. installation testing
   e. performance testing

21. A function X in component A requires a pointer to an integer to be passed as an argument, but a call to the function X in component B passes the value of an integer instead. Which type of testing would most likely expose this defect?
   a. unit testing of component A
   b. integration testing of components A and B
   c. performance testing
   d. installation testing
   e. acceptance testing

22. The gas pump subsystem is supposed to allow the user to choose whether or not a receipt is printed, but the print function has not been implemented. Which type of testing would most likely expose this defect?
   a. unit testing
   b. integration testing
   c. performance testing
   d. acceptance testing
   e. function testing

23. A configuration file used by the reporting subsystem is not placed in the correct directory in the customer's environment. Which type of testing would most likely expose this defect?
   a. integration testing
   b. installation testing
   c. performance testing
   d. acceptance testing
   e. function testing

24. A change is made to correct a fault. The fault has been fixed, but it has caused a fault in previously functioning code. Which type of testing would most likely expose this defect?
   a. unit testing
   b. acceptance testing
   c. regression testing
   d. performance testing
   e. installation testing

25. The customer is unhappy with the number of screens that must be traversed before getting to the parts list screen, a screen accessed frequently when using the system. Which type of testing would most likely expose this defect?
    a. integration testing
    b. installation testing
    c. performance testing
    d. acceptance testing
    e. function testing

26. The GSCS system includes reused components from a third party vendor. The source code for the reused components is not available. Which type of testing is feasible?
    a. all-paths
    b. def-use
    c. branch testing
    d. black-box testing

Tom, the manager in charge of testing for the GSCS, is concerned about the reliability of the system. He decides to seed faults in the code to estimate the remaining faults. He has two teams testing the code. One team is led by David. The other test team is led by Daniel. Suppose 50 faults have been seeded in the code.

During testing by David's team, 70 faults are detected. Forty of the detected faults are seeded faults.

27. What is the Mills estimate for the percentage of remaining, non-seeded (indigenous) faults in the code?
    a. 10%
    b. 20%
    c. 50%
    d. 80%
    e. It is impossible to determine from the information given.

28. What is the Mills estimate of the total number of indigenous faults remaining?
    a. 7.5
    b. 10
    c. 17.5
    d. 30
    e. 37.5
    f. It is impossible to determine from the information given.

Suppose the same code is given to Daniel's test team. His team finds a total of 50 faults. 35 of the faults found by Daniel's team were also found by David's team.

29. Using the numbers for Daniel's team, what is the Mills estimate for the total number of indigenous faults remaining?
    a. 17.5
    b. 30
    c. 50
    d. 71.5
    e. It is impossible to determine from the information given.

30. What is the effectiveness of David's group?
    a. 30%
    b. 50%
    c. 70%
    d. 85%
    e. It is impossible to determine from the information given.

31. What is the effectiveness of Daniel's group?
    a. 30%
    b. 50%
    c. 70%
    d. 85%
    e. It is impossible to determine from the information given.

32. Using the data from both test groups, what is the estimate for the total number of faults?
    a. 100
    b. 87.5
    c. 70
    d. 50
    e. It is impossible to determine from the information given.

33. Suppose 39 faults have been seeded into a component. Testing of the component has uncovered 32 of the seeded faults without uncovering any additional non-seeded faults. What is the level of confidence that the component is fault-free?
    a. 78%
    b. 80%
    c. 82%
    d. 86%
    e. None of the above.

34. Using the data from the previous question, how many of the seeded faults would have to be found without uncovering additional indigenous faults to have a 90% confidence level?
    a. 32
    b. 35
    c. 36
    d. 37
    e. None of the above.

35. Consider the following excerpts from problem reports filed for the GSCS. In which type of report, discrepancy or fault, does each item belong? Answer fault report or discrepancy report.
    a. "A segmentation violation occurred while viewing the part list. The part list array may not be big enough to hold all parts. Check the PartList class header."
    b. "In the requirements document, Section 2.1.5, a Print option should be included in all File menus. The File menu for the Part Configuration Screen does not include a Print option."
    c. "After submitting the Add Part form, it took 3 minutes before the results came back. Submitting the form should not take more than 1 minute."
    d. "When the cashier list is displayed, the newly added cashiers do not appear on the list. Check the Add method in the CashierList class. There was a similar problem with the part list. Theresa worked on

the part list problem. See report number 201 for more details on the problem and her solution.”

Review Exam 3 Answers

1. f; Readable code does not always improve efficiency. Sometimes, there is a tradeoff between readability and efficiency. [Section 7.1, 7.2]
2. b; Passing a variable by reference means that the value of the variable will be changed. If the altered value is not needed after the function terminates and the variable size is small, the variable should be passed by value.
3. FALSE; It is not safe to assume that the function will work properly because it was tested only with integers. The function may produce the wrong output for non-integers, or fail due to round-off errors.
4. a; only one file can be read and printed with this function. It could be written more generally.
5. b; The two loops can be combined to make this code more efficient.
6. b; Since the sales array of the Cashier structure is very large, the argument to SumSales should be passed by reference to improve efficiency.
7. e
8. c; Formatting of this code makes it misleading. It hides a fault in the code.
9. c; Because n is an integer, the expression 2/(n-1) will evaluate to an integer giving an incorrect answer. [Section 8.1]
10. e; The code doesn't do what the comment describes. The variable i is initialized incorrectly. [Section 8.1]
11. c; list[10] is out of the defined array boundary [Section 8.1]
12. a; The first element of the array (list[0]) is overwritten during the first iteration of the loop. The overwritten value is used in future iterations. When the loop terminates each element of the array will be 0. The computation does not check for xrange = 0 which may lead to a division by zero error. Both of these faults are computation faults. [Section 8.1]
13. a; [Section 8.3]
14. b; With the bottom-up and big-bang approaches, the whole system has to be built before a working program can be shown to the customer. With top-down testing, stubs and drivers can be used to test the system before the entire system is built. [Section 8.4]
15. d; big-bang testing [Section 8.4]
16. a; top-down testing [Section 8.4]
17. b; bottom-up testing [Section 8.4]
18. e; Modified top-down testing [Section 8.4]
19. c; sandwich testing [Section 8.4]
20. a; unit testing; This defect can be isolated to a single function in a single component. Unit testing should uncover this type of defect. [Section 8.2]
21. b; Since this defect involves the interface between the two components, integration testing of components A and B should detect the defect. Unit testing of component A would not uncover the defect since the defect exists in component B. [Section 8.2]
22. e; Function testing is used to determine if the functions described in the requirements specification are actually implemented in the system. [Section 8.2]
23. b; The purpose of installation testing is to make sure that the system will function properly where it is installed. [Section 8.2]
24. c; The purpose of regression testing is to ensure that changes to the system have not negated the effects of previous tests. [Section 9.1]
25. d; Acceptance testing is where the system is checked against the customer's requirements. [Section 8.2]

26. d; Because the code is not available, the structure of the code is not available for testing. In this case, black-box testing is the only feasible option. [Section 8.2]
27. b; The percentage of indigenous faults remaining is equal to the percentage of seeded faults remaining. (1 - 40/50) = .2 [Section 8.8]
28. a;

$$\frac{seeded\_faults\_found}{seeded\_faults} = \frac{indigenous\_faults\_found}{indigenous\_faults}$$

$$indigenous\_faults = \frac{seeded\_faults \times indigenous\_faults\_found}{seeded\_faults\_found}$$

$$indigenous\_faults = \frac{50 \times 30}{40} = 37.5$$

indigenous_faults_remaining = 37.5 - 30 = 7.5
[Section 8.8]

29. e; The number of seeded faults for the second test group is not given. [Section 8.8]
30. c; effectiveness = overlapping faults/faults found by the second group effectiveness = 35/50 = 70% [Section 8.8]
31. b; effectiveness = 35/70 = 50% [Section 8.8]
32. a; total faults = 35/(.7 * .5) = 100 [Section 8.8]
33. b;

$$C = \frac{\binom{39}{31}}{\binom{40}{32}} = 0.8$$

[Section 8.8]

34. c;

$$C = \frac{\binom{39}{31}}{\binom{40}{s}} = 0.9$$

$$0.9 = \frac{s}{40}$$

$$s = 36$$

[Section 8.8]

35.
   a. fault report; The description of the problem includes information from the developer's point of view. [Section 9.8]
   b. discrepancy report; This description describes a difference between the requirements and the implementation. [Section 9.8]

c. discrepancy report; This description describes a problem from the user's point of view. [Section 9.8]
d. fault report; The description of the problem includes information from the developer's point of view. [Section 9.8]

Chapter 11: Maintaining the System

Learning Objectives:
After studying this chapter, you should be able to:

- Define what is meant by system evolution, and understand how it affects the software development process.
- Define what is meant by a legacy system, and understand how its characteristics affect maintainence.
- Define impact analysis, and understand when, how, and why it is done.
- Describe software rejuvenation, and why it is necessary.

Summary:
Delivery of a system to the customer does not mark the end of the software developers' involvement with the system. Rather, many systems require continuous change, extending even past delivery. In general, the more closely a system is tied to the real world, the more likely it will be to require changes (and the more difficult those changes will be to make). Software maintenance deals with managing change in this part of the life-cycle.

Performing maintenance requires its own set of skills, in addition to those required for software development. Maintainers interact continually with colleagues, customers, and users in order to effectively define problems and find their causes. Maintainers need to be good detectives, testing software thoroughly and hunting down the sources of failure. Maintainers also need to understand the "big picture" of how software systems, with many complex interactions among their components, interoperate with the environment. Impact analysis, which builds and tracks links among the requirements, design, code and test cases, is necessary to evaluate the effects of a change in one component on the rest of the system.

Another important technique is software rejuvenation, which involves the redocumenting, restructuring, reverse engineering and reengineering of an existing system. The overall goal is to make hidden information explicit, so that it can be used to improve the design and structure of the code. Although complete rejuvenation is unlikely in the near future, it is being used successfully in domains that are mature and well-understood, like information technology.

Measuring maintainability is difficult. A true measure of maintainability requires evaluating the external behavior of a system and tracking the mean time between failures. However, waiting until the system fails is too late to be of much use to developers and maintainers. Instead, internal attributes of the code, such as size and structure, are used to predict those parts of a software system that are likely to fail, based on past history. Static code analyzers are tools that aim to assist in this identification process.

Exercises:

1. The "Millenium Bug" or "Y2K problem" is perhaps the most infamous software maintenance problem. Many computer systems represent the year as only two digits and are expected to have problems in the year 2000, when the value for the new year ("00") is suddenly less rather than greater than the value for the previous year ("99"). Find a discussion of the Y2K problem written for nonscientists, for example, in a newspaper or popular magazine. How many of the maintenance problems listed in section 11.3 are

accurately presented in the article? Are there issues in section 11.3 that contribute to the Y2K problem but are not given in the article?

2. The Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center collects data from all phases of its software development projects. When users fill out failure reports, they are asked to indicate the severity of the defect according to the following scale: major defect with no workaround; major defect, but workaround exists; cosmetic defect. How can this information be used to help the SEL understand its maintenance process better?
3. Maintenance is an area of great interest to software engineering researchers. Conferences and workshops such as ICSM (the International Conference on Software Maintenance) and WESS (the Workshop on Empirical Studies of Software maintenance) are devoted exclusively to maintenance issues, as is the Journal of Software Maintenance. Review a recent conference proceedings or journal issue and summarize the types of problems maintenance research addresses.
4. Researchers with the Institute for Information Technology of the National Research Council, Canada, study maintenance by observing the work practices of software engineers who are engaged in maintenance projects. A paper by Janice Singer and Timothy Lethbridge summarizes the methods they use to collect this type of data. (J. Singer, T. Lethbridge (1996). “Methods for Studying Maintenance Activities.” In Proceedings of the International Workshop on Empirical Studies of Software Maintenance, Monterey, CA. Also available at http://wwwsel.iit.nrc.ca/projects/easse/). Summarize this paper from the viewpoint of a software maintainer. How disruptive are the data collection methods likely to be to the maintainer's work practices? What does the maintainer stand to gain by participating in such a study?
5. Revisit the program you wrote for exercise 1, Chapter 7. Change the underlying data structure of the stack to a linked list rather than an array, and the data type of the stack elements to a string rather than an integer. How hard was this to do? On what types of activities did you spend your time? Critique your earlier program in terms of maintenance effort, paying attention to ideas such as comments, modularity, encapsulation, and others that affected the ease or difficulty of this task.

Answer Guidelines:

1. Many of the issues in section 11.3 relate in some way to the Y2K problem. Some examples: The limits of human understanding are certainly applicable. There is a definite limit to how quickly maintainers can approach a system that is unfamiliar to them and understand enough about it to make the correct changes for a maintenance problem. That difficulty is compounded when the system being maintained is old and the chances of missing documentation or even source code have increased. Management priorities have been a major contribution to the problem. Since Y2K maintenance does not result in a new product but rather keeps an old product running, management in many cases did not assign a high priority to maintenance in general and Y2K maintenance in particular. As a result, Y2K was often not a high priority until very close to the year 2000, when the problem was no longer avoidable. Morale has been a problem in some cases, in which software practitioners were assigned part-time to handle the Y2K problem in addition to their other duties. This type of situation tends to reinforce the belief that Y2K maintenance is not an important or interesting task.
2. The severity scale helps the SEL understand better how its development process affects maintenance. It gives more information than simply

collecting the number of changes that have to be made during maintenance; it allows some insight into whether most of the changes that have to be made are mostly small changes or large redesigns.

3. Common categories of software maintenance research include: program understanding, predicting effort, predicting components likely to require rework tool support.
4. By understanding the techniques that maintainers find useful in practice, this research hopes to provide a better idea of how software maintainers can benefit from tool support. That is, the point of this research is that tools should be created after the tool developers understand for which tasks maintainers really need support. The methods listed in this paper are at varying levels of intrusiveness; the authors understand that less intrusive means for collecting data will be more welcome by maintainers. By participating in such a study, however, maintainers can expect that tools will be created that better address the requirements of the job they are undertaking.
5. Answers will vary depending on the quality of the original program. The exercise will be more useful if you have forgotten the details of the program since it was written, since then you will have to rely on reading the code, comments, and documentation. This situation is similar to that software maintainers face when working on code they themselves did not originally develop, or developed some time ago. It is hoped that you will find that this exercise to be easy if you have made the code well documented, straightforward and easy to understand, and modular. However, some of these factors may be more important than others for your program, and other factors may also be a consideration.

Chapter 12: Evaluating Products, Processes and Resources

Learning Objectives:
After studying this chapter, you should be able to:

- Discuss how feature analysis, case studies, surveys and controlled experiments differ, and the circumstances under which each is appropriate.
- Define measurement and validation, and understand how they are carried out in software development.
- Describe the Capability Maturity Model, ISO 9000 and other process models, and the differences and similarities between them.
- Describe what is meant by people maturity, and the role this may play in a software organization.
- Describe how and why development artifacts are evaluated.
- Define return on investment and its importance with respect to the software development process.

Summary:
Previous chapters have given an overview of the large variety of methods and tools that are available for use by software developers, throughout the software life-cycle. This chapter takes up the question of how developers can decide which method or tool is best to use. Answering this larger question requires accurate answers to a number of more specific questions:

- How can developers evaluate the effectiveness and efficiency of what they are already doing, so that they can tell if a change to the development process actually results in improvement?
- For a given situation, how can developers know which is the most appropriate method or tool to introduce into their development process?
- Once a change has been made, how can developers demonstrate that the products, processes and resources have the desired characteristics (such as quality)?

Evaluation of software development requires first choosing whether the most appropriate type of study is a feature analysis, survey, case study, or formal experiment. Models and frameworks are necessary to help developers understand the relationships being investigated; of course, the models and frameworks themselves must be evaluated in terms of how closely they match what is already known. Regardless of the type of study, measurement is essential for any evaluation. It is important to keep in mind that measures must be validated, that is, it must be shown that measures actually capture the concept of interest, and that the resulting predictions are accurate. A second important concept that is important to keep in mind is the difference between assessment and prediction. These common principles should be applied to the evaluation of software products, processes and resources.

Product evaluation is usually based on a model of the attribute of interest. This chapter introduced three quality models and discussed how each one addresses particular concerns about how specific attributes combine to form a picture of quality as a whole. Other considerations, such as software reuse, imply their own sets of product attributes that must be evaluated.

Process evaluation can be done in many ways. Post-mortem analysis looks back at completed processes to assess the root causes of things that went wrong. Process models, such as the Capability Maturity Model, SPICE and ISO 9000, are useful for assessing the amount of insight into, and control over, the processes being used.

The CMM has inspired a host of other maturity models, including a people maturity model to assess the degree to which individuals and teams are given the resources and freedom they need to do their best. Software projects require other types of investment as well, including money and time. Return-on-investment strategies can indicate whether business is benefiting from investment in people, tools and technology.

Exercises:

1. In the key references section of this chapter, it is noted that the journal Empirical Software Engineering publishes not only descriptions of empirical studies, but data from these studies as well. What do you think are some of the benefits to other researchers of having access to the data? Are there any benefits to practitioners?
2. Give an example, from a previous programming project, of when you engaged in black-box reuse. What are some of the benefits that can be expected from black-box reuse? What are some drawbacks? Give an example of a system for which black-box reuse would not have been appropriate.
3. Take a look at the latest issue of a journal that presents articles about software engineering. (IEEE Transactions on Software Engineering, the Journal of Systems and Software, and IEEE Computer are good examples.) Of the articles that present a new technique or tool for software development, how many actually present some kind of evidence that the proposed technique is an improvement over what is currently used? For those that do, classify the type of empirical study used, and identify the variables.
4. Select one of the studies that you identified in the answer to question 3. Analyze this study with respect to the common pitfalls in evaluation that are described in Table 12.2. For each pitfall, assess whether or not the study has successfully avoided the problem, and explain your reasoning. If the article contains sufficient information to allow you to judge whether the pitfall was avoided, that should also be noted. If there are pitfalls in this study, do the authors identify them and discuss their impact on the results?
5. A paper by Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger addresses in some detail common pitfalls of case studies in industrial environments (B. Kitchenham, L. Pickard, S. L. Pfleeger (1995). “Case studies in method and tool evaluation.” IEEE Software, 12(4): 52-62). Use this paper to critique a case study of some software engineering technology. (The journals suggested in question 3 are good sources of case studies.) If there are problems with the study, do you think they can be corrected in such a way thatthe study will still be feasible to run?

Answer Guidelines:

1. Perhaps the most important benefit of published data is that they help researchers check each others' conclusions; they allow researchers to analyze the same data and see if their results match. Publishing data also assists in comparing data among studies since it allows researchers to understand any desired attributes of data sets (e.g. mean, median, amount of variation among the values). In the same way, publishing data also helps practitioners better understand the results of applying a technology, and may allow them to compare results in their own environment to data from outside.
2. Although you may report experiences with black-box code libraries or other forms of reuse, almost every programming language has the option to

include functions from standard libraries, which may also be an example of black-box reuse if the source code is not available. Benefits include being able to save effort by reusing functionality rather than implementing it from scratch; testing effort is also saved since presumably the reused component does not have to undergo unit testing. Drawbacks include the time required to find the component and figure out how to configure it for use in a particular system. A drawback unique to black-box testing is the fact that, since the internals of the component cannot be tested, it is more likely that defects in the code will be propagated unnoticed to other systems. Systems in which reliability or safety is an overriding concern may not be good candidates for black-box reuse for this reason.

3. Answers will vary. The point of the question is to determine whether you can differentiate articles describing empirical studies of the kind mentioned in this chapter (section 12.1) from articles in which the claims are not substantiated, or are substantiated only with analytical reasoning. It is important to be clear as to which category each article you review falls into, and to back up your categorization with points from the article.
4. Answers will vary. The point of the question is to assess whether you understand the meaning of the nine pitfalls listed in Table 12-2, and the form they take in evaluation studies. Make sure you understand the definition of each pitfall, and can answer whether or not it appears in the study.
5. Answers will vary, depending on the case study selected. There are many guidelines in the Kitchenham et al. paper that can be used to critique case studies; your paper should address questions such as whether a case study was an appropriate form for this study in the first place, and whether there are any problems of construct, internal, or external validity. In many cases, correcting defects in empirical studies may require an infeasibly large amount of time or effort from the developers who serve as subjects; you should consider whether this is true for the study you have chosen.

Chapter 13: Improving Predictions, Products, Processes and Resources

Learning Objectives:
After studying this chapter, you should be able to:

- Discuss strategies for improving predictions. Explain how reuse and inspections can be used to improve software products.
- Describe how Cleanroom and maturity models can be used to improve software processes.
- Describe how investigating trade-offs is necessary to improve software resources.

Summary:
Chapter 12 provided an introduction to the methods used for evaluating software products, processes, and resources to determine their impact on development and maintenance. This chapter provides concrete examples of software evaluation and improvement by discussing actual instances of technology adoption in four areas: prediction, products, processes and resources.

Predictions can be improved by using u-plots, prequential likelihood and recalibration to reduce noise and bias. Products can be improved as part of a reuse program, or by instituting an inspection process. Processes can be improved by evaluating their effects and determining relationships that lead to increased quality or productivity. For example, models can be developed, based on past history, to predict when components will be faulty; this technique reduces the effort required to maintain a system, and ultimately leads to higher-quality software. Similarly, process maturity frameworks may assist organizations in implementing activities that are likely to improve software quality, although careful controlled studies have not yet provided sufficient evidence as to their effectiveness. Finally, there is promise of improvement in resource allocation as we learn more about human variability and examine the trade-offs between effort and schedule.

One of the common threads in the technologies discussed in this chapter is the importance of human factors research. Many of the studies reported in this chapter emphasize the need for teams to check each other's work. Inspections, Cleanroom, reuse and other quality-related processes involve the careful scrutiny of one person or organization's work by another. All of these approaches are largely dependent on people factors in order to be effective. In general, researchers admit that human variability is a key factor in determining whether quality and schedule goals will be met. Thus, an especially promising area of research in software engineering is into issues such as team size, collaboration styles, and good working environments, which determine how software engineers themselves can best be supported. A promising way to improve this type of research in software engineering is to learn from similar studies that have already been undertaken in the social sciences.

Research on improvement issues is growing, as developers increasingly ask for empirical proof that proposed technologies really work. This chapter illustrates the need for more surveys, case studies and experiments; the Basili and Green example shows how a collection of studies can be organized to build on each other. Of course, to be carried out effectively, such studies require that developers are willing to participate in case studies and experiments and to give feedback to those who are trying to determine what leads to improvement.

Exercises:

1. An organization currently uses informal, English-language requirements and requirements reviews in its software development process. A consultant has recommended that it switch to the more formal requirements language, Z. The organization decides to try out Z on a new project, to see whether or not it improves the software process. To evaluate Z with respect to the current process, what types of measurements should be collectedon projects using English requirements? On the trial project? Justify your answers.
2. Describe an empirical study that could be used to assess whether Z represents an improvement for the organization. What type of empirical study would you select, and why? How much confidence could the organization have in the result? How much disruption would be necessary to the organization's usual software development process?
3. In a 1997 paper, Vic Basili describes a series of studies of a particular kind of software technology, called software reading. (V. Basili (1997). "Evolving and packaging reading techniques." Journal of Systems and Software, 38: 3 - 12.) Each study in the series contributed some knowledge about the use of this technology in a particular environment. A number of different types of studies were used: Some studies looked at whether or not the technology was feasible in the environment, other studies tested very specific hypotheses about the technology, and still other studies examined the use of the technology in detail. What kind of studies would you recommend for each of those goals? Sketch a series of studies for the organization interested in Z (discussed in questions 1 and 2) that incorporates all three goals.
4. Many studies in computer science compare different technologies and do not involve human factors. For example, a study of a new algorithm may seek to determine whether it runs faster than an older version on practical data sets. However, many empirical studies in software engineering involve human subjects, because they need to assess the usefulness of development techniques for the people who will use them. Find a recent journal article that describes an empirical software study using human subjects. Briefly summarize (2 paragraphs) the study and itsresults. What are the things that make studies using human subjects different from studies that do not? Use specific examples from the journal article to illustrate your points.
5. Do a search of the literature in which you identify the relevant papers on two different approaches to process improvement. Use the references given in the textbook as a starting point. Compare and contrast the two approaches in a short report (less than 5 pages). Your report should be a summary of the two approaches, written for an organization thinking of investing in process improvement. You should answer questions such as: Where have the approaches already been applied? Have they been shown to work? What support is required from the organization? Identify other relevant criteria as you see fit.

Answer Guidelines:

1. The organization will need to collect measures of how effective its requirement process is; measures such as the time and effort required from developers would be good choices, as would some measure of the quality of the resulting requirements. (You should remember that "quality" is a difficult concept to measure directly, and propose a way it can be feasibly assessed. Measures of quality may vary depending on the interests of the organization, so you should be sure to justify your answer.) Other variables are required to describe the context in which the process is applied; for example, the type of project or experience of the developers

using it. Collecting the same type of measures for both the Z and natural language requirements processes will enable comparisons between the two.

2. You may choose feature analysis, case studies or controlled experiments (surveys are excluded since there are no retrospective data). Because multiple answers are possible, you should be sure to justify your choice: What do you think are likely goals for the organization conducting the study? Which type of study stands the best chance of achieving those goals? You will also need to correctly answer the follow-up questions based on your choice of study type. Confidence and disruption are generally directly related; feature analyses would produce low confidence but minimal disruption, while controlled experiments would yield high confidence but place the most extra demands on developers' schedules.
3. A feature analysis or controlled experiment could provide a quick answer as to the feasibility of a technology. A controlled experiment is best for testing a particular hypothesis, since variables besides the one of interest can be controlled. A case study is probably the best choice for getting a more in-depth knowledge, since a project can be followed all the way to completion (and if a sister project can be found, compared to a similar project not using the technology to understand its effects). Answers as to the series of studies will vary, but here is one possibility: A feature analysis is undertaken to determine if Z looks promising. It seems to match the needs of the users, so volunteers are solicited who would be willing to try out the technology and report on the results. These volunteers are assigned to a new project, which is monitored as a case study. A comparison with a similar, previous study on which Z was not used seems to indicate that the use of the technology represents an improvement in the way requirements are specified. Finally, a larger controlled experiment is conducted to see if the improvement is noticed for a wide range of the developers in the organization.
4. Answers will vary depending on the studies selected. One acceptable answer could be sketched in the following way: Studies with human subjects have to contend with a wide variation among subjects, even those with similar backgrounds and experience levels. It is rare to find subjects who perform equally well on all tasks, even if they have had similar experience or training; humans have natural aptitudes and interests. Studies with human subjects also have to contend with variation within subjects; that is, humans do not perform the same task at a consistent level. They have bad days, or learn things as they go along; they can be distracted, or focus more intently on the task. Studies of computer technologies can be expected to produce much more deterministic results.
5. Answers will vary. In identifying the relevant literature on a particular approach to software improvement, you should focus first on finding the published work in which the approach is originally defined. If there have been major changes to the approach since it was first published, you should try to track down literature in which the changes are proposed and discussed as well. Also, you should look for publications that describe how the approach has been applied in practice - the most recent publications and most thorough descriptions are always among the most relevant. Use this list of publications to support the points you make as you compare and contrast the two approaches. Begin by summarizing the definitions of the approaches. Then, summarize the publications describing their application. Were you able to find many publications describing the use of the approach in practice? If not, has your search been less than thorough, or is the approach simply not used often? In what types of organizations have the approaches been applied? What kind of results have been obtained? Can you say anything about the factors that are present in

each case that may have contributed to the good or bad results that were seen?

Chapter 14: The Future of Software Engineering

Learning Objectives:
After studying this chapter, you should be able to:

- Describe where the field of software engineering stands with respect to Wasserman's eight steps.
- Describe what is meant by "technology transfer," and why it is important.
- Understand what kinds of evidence can bear on technology adoption, and how researchers provide such evidence.
- Understand how decision making can (and should) occur in software project management.
- Describe some important areas for future work in software research and practice.

Summary:
Software engineering is a young field (the term itself was first used in 1968) but has already seen great changes. The field has progressed with the development of complex programming languages and more reusable products. Formal methods for problem description, tools for assisting software development, and useful design principles have been developed and helped software engineers tackle ever-larger problems. However, there is more accuracy in the large than in the small; the field tends to agree on broad principles but is less successful in pinning down the effects of specific decisions that project managers will need to make.

In the terms of Wasserman's eight principles, software engineering has experienced:

- The use of *abstraction,* to help focus on the core of the problem, most successfully applied in design and code. However, more work is needed in other areas such as software requirements, work habits, and user profiles.
- The development of a wide range of *analysis and design methods and notations* to suit personal preference and comfort. However, no common method or notation has been developed that the others can be mapped to, to simplify communication and understanding.
- The role of *user interface prototyping* become more and more critical. Work needs to continue in this area to support the production of ever more responsive and useful products.
- The very beginning of the identification of *architectural styles and patterns* with their associated pros and cons.
- A growing understanding that *software process* affects product quality, but not exactly how that quality is affected by the visibility and controllability of the process. More work is needed into how specific process choices affect the development of the product.
- A focus on *reuse*, mainly of code. Reuse must be expanded to other work products throughout development and maintenance.
- The use of *measurement* to see if products meet quality criteria. Future work needs to expand to measure key characteristics of products, processes, and resources in ways that are unobtrusive, useful, and timely.
- Significant investment in *tools and integrated environments* that have not lived up to their promise. Current and ongoing efforts are looking at tools with more realistic expectations, for feasible tasks such as

tracing connections among products, background measurement, and reuse support.

An important area where improvement is needed is technology transfer, that is, the transformation of a promising research idea into a technology that is useful and effective for practitioners. Technology transfer decisions have both a technical aspect (finding the right technology to solve a problem) and a commercial aspect (appealing to customers who need to have the problem solved). Widespread commercial adoption of promising technologies can take a decade or two, so given the time-to-market pressures of the industry today it is not surprising that software development organizations often rush to grab new technologies before there is clear evidence of benefit. Enabling decisions to be made on the basis of better and clearer evidence is a primary goal in the improvement of technology transfer practices.

Looking to the area of “diffusion research” in marketing helps us understand how decisions about technology adoption are made. Data across many organizations show that there are distinct types of technology adopters, who exhibit varying degrees of willingness to try out a new technology: innovators, early adopters, early majority, late majority, and laggards. Each group has different requirements for evidence of a technology’s effectiveness and the level of support they need before they are willing to invest in using it. Knowing and being able to address these requirements is thus an important prerequisite to seeing new technologies effectively adopted in industry. However, studies have shown that software researchers have their own goals and preferences for the kinds of evidence they collect, which don’t always address the needs of practitioners. This mismatch between the two communities diminishes the relevance of research work and results in technology decisions being made without the kinds of evidence that are really needed.

Conclusions about a technology have to be drawn from a collection of evidence, where each piece of evidence might count more or less than others based on what is known about it. The legal community has a long tradition of building conclusions in this way and can provide some guidance for addressing important issues. For example, we can place a particular piece of technology into one of five categories, based on what is known about its source and credibility: tangible evidence, testimonial evidence, equivocal testimonial evidence, missing evidence, or accepted facts. When various pieces of evidence conflict, decision makers need to decide whether some piece of evidence is flawed, or whether the information can be used to refine the conclusion by understanding how variations in the context from which the evidence was collected affected the results. In software technology adoption people generally look for evidence about a technology’s relative advantage, compatibility, complexity, tailorability, and observability.

All of this information can be used to outline a general process for technology transfer. First, there should be a preliminary evaluation of a technology within an organization’s culture. The results contribute to a growing body of evidence that can be evaluated itself to see if it contributes compelling evidence for adopting the technology. If the decision is made to invest in the technology, then effort must be spent to package and support it, to facilitate its adoption throughout the organization.

Of course, software development involves decision-making on a wide range of issues, not just technology adoption. Again, however, many other fields contribute both descriptive and predictive theories that help us understand the process. One such theory identifies four important elements that affect any decision: problem finding, problem context, problem solving, and legitimization. Group decision-making adds even more complexity, since issues of trust, communication, and cooperation are added to the mix. Group issues can be addressed by selecting an appropriate decision strategy, e.g., a dialectic process, third-party reconciliation, brainstorming, round robin. In an organization, the right choice of strategy can also depend on whether the decision is strategic, tactical, or routine.

Observational studies of decision-makers at work have led to a “recognition-primed” decision model, which suggests that people keep a mental repository of past experiences that can be compared to the current situation. In this model, people reason about which experience is closest to the current situation, and then use mental simulation to estimate whether the same solution can apply. However, the reasoning process is not always so straightforward. Bias can creep into decision-making in numerous ways: examples are contextual bias, stereotype threat, status-quo bias, and a reluctance to appear negative. People often tend to over value evidence that is case-specific, recent, or particularly vivid.

The situation of group decision-making is similar. Techniques such as Delphi exist to help teams converge to a solution. However, bias can enter the process, often through issues of group dynamics.

At this time, the field of software engineering is grappling with not only technical issues, but also with questions about the field itself. Can software engineers be licensed and trained in the same way as other engineers? What material belongs in a comprehensive software engineering “body of knowledge”? One of the themes of this chapter (and of this book) has been that, to address such questions, we need to view software engineering in its broader setting, recognizing that software is the product of creative people working in teams. We must study the ways we are similar to other engineers but also embrace other disciplines, including the social sciences, so that our processes are effectively tailored to the human beings applying them, and our products are as useful as possible to our customers.

Exercises:

1. The “recognition-primed” decision model postulates that people make decisions by keeping a repository of past decisions and their results, against which the current situation is compared to suggest a likely strategy. Ongoing work is attempting to apply this theory at the level of whole organizations, by creating an organizational “experience base,” describing the past work of all employees, which can be searched for answers to new problems. Can this model be applied directly to the organization? What are some complications that will have to be overcome before systems of this type could be effective?
2. Find a paper from the research literature describing a new technology. Remember in this context that “technology” can have a broad meaning, including software processes, tools, or specific techniques. Describe the particular technology being proposed. Does the paper describe who the anticipated users of the technology will be? Can you categorize those users according to the types of adopters in Figure 14.1? What arguments are made as to the technology’s effectiveness? How could you classify those arguments according to the categories of evidence proposed by Schum (described in section 14.2)?
3. Find a paper describing an organization’s practical experience with a particular technology. If at all possible, find one describing experience with the same technology you chose in question 2. (Many conferences, including ICSE, now include tracks dedicated to industry reports, making

these conference proceedings a good place to start your search.) How does the paper measure the success (or lack of success) with the technology? What kinds of evidence are presented to support the evaluation of effectiveness? How would you classify that evidence according to the categories of evidence proposed by Schum (described in section 14.2)?

4. As a project manager, John has been using design inspections on his projects since he came to his current employer, even though his manager does not believe that the inspection personnel have the necessary background to be successful. Over the last five years, seven design inspections were undertaken and John considered all of them successful (i.e. they all found some significant issues, and no major defects caused by design were found in the product in later development or use). However, in this year three inspections were undertaken and all were unsuccessful (major problems slipped through). One of these instances was particularly embarrassing for John since a major defect was not found until after delivery to the customer, and the redesign that was necessary was particularly expensive. Now John is starting a new project (which is not very similar to his previous projects in this organization) and has to consider whether to spend the resources on a design inspection again. What sources of bias should he be aware of that might affect his decision?
5. Given the situation described in question 4, the body of evidence John is accumulating about design inspections has internal contradictions. What can John do to try to draw a meaningful and accurate conclusion?

Answer Guidelines

1. It is hard to apply the model directly, primarily for two reasons: first, the information at the organizational level comes from many employees, not just one person’s past experience. Secondly, the information has to be made explicit, so that other people can share and understand, not just stored for personal use. From these differences, there are a number of difficulties that arise. Some of the most important are: Each item that is stored has to be classified in some meaningful way so that it can be found again, and the user can judge how close it is to the current situation. A related problem is that queries to the experience base have to satisfy two (often contradictory) requirements: they must make sense to the person doing the query, matching how he or she actually thinks of the problem trying to be addressed, and also be powerful and accurate enough to be used to find relevant information in the base. And, the information has to be stored in such a way that it is useful to the person doing the query when it is found. That means that somehow the context of the decision, the decision itself, and the results have to be described in a way that is understandable and meaningful.
2. Answers will vary depending on the paper chosen. It is important to note that research papers often fail to address the anticipated users of a proposed technology; the type of adopter for whom this technology is suitable may have to be inferred from the amount of existing evidence and support for the technology. The type of evidence cited to support the effectiveness of software technologies vary widely, although there is some evidence that the number of papers citing some empirical evidence (which would be categorized as testimonial evidence in Schum’s taxonomy) has been increasing.
3. Answers will vary depending on the paper chosen. Ideally, measures of success would come from Rogers’ list of technology attributes described in Section 14.2. The type of evidence cited to support the effectiveness of software technologies vary widely.

4. In this specific situation, a number of biases might come into play for John. Decision makers tend to be biased toward evidence that is recent and vivid, so John might over value the evidence from this year, especially from the project that he found particularly embarrassing, and make an overly negative judgment about the inspections’ value. Since people also tend to be biased toward case-specific evidence, John might be tempted to make a larger generalization about inspection effectiveness even before he examines how the new project differs from the old ones in ways that might impact the inspections. There is nothing in the question to lead us to believe that contextual bias would apply, although how John frames the question to himself would certainly play a role. If he asks himself whether he “wants to spend the resources” or “wants to actively pursue a higher-quality design” in the new project (two different ways of saying the same thing), his answer could certainly be influenced. There would not appear to be any status quo bias in this example since it is clear that the new project will not be a familiar environment. Although it probably wouldn’t come into play in the decision, stereotype threat may be affecting the results; if the inspection personnel know that others in the company believe they don’t have the background to be effective, this could be influencing their performance.
5. Since all of the evidence comes from personal experience, John shouldn’t weight some of it less than others due to the source (for example, he might not have paid as much attention to some inspections if he had only heard about them second-hand). John should examine the evidence to make sure that what he knows is accurate. Were the previous inspections really as successful as he thinks? Did the ones this year really miss issues they should have been expected to catch? If the evidence is accurate then he should pursue contextual factors that might explain the discrepancy, for example, were the inspections this year on different types of systems, using different personnel, or using a different inspection process? These contextual factors might explain why some were successful and others not.

Review Exam 4

The following questions are in reference to a hypothetical “Gas Station Control System” (or GSCS) that will be used to help manage an American-style gasoline or service station. Our hypothetical gas station basically provides two services:

- There is a small store that carries car parts. Inside the store is at least one cash register, operated by a cashier who is an employee of the gas station.
- There are a number of gas pumps, at which customers can park their cars, interact with the system to pay via credit card, and then pump their own gas. Alternatively, the customer can pay for his or her gas via cash or credit card by going into the store and paying directly to the cashier.

Thus the GSCS has two main classes of users. The first is the cashier, who uses the GSCS to record purchases of car parts by customers. The GSCS must allow the cashier to enter the type and number of parts purchased, then compute the total purchase price and handle the payment. Customers purchasing gasoline are the second type of user. These customers interface with the system at the gas pump, by specifying the amount and type of gas they will buy, paying either at the pump or to the cashier, and then pumping the gas themselves.

The system also has to interact with other automated systems to perform its tasks. For example, in order to accept credit card payments, the GSCS must interface with a system maintained by the credit card company. The credit card system is responsible for checking that the customer's account is in good standing and can accommodate the amount of the purchase, and for debiting the customer's account and eventually reimbursing the gas station. The operation of these external systems is beyond the scope of the GSCS, although the GSCS needs to know how the external systems will communicate the success or failure of their tasks.

The GSCS is divided into three subsystems:

- A gas purchase subsystem, that takes care of customer interaction with the gas pumps;
- A cashier subsystem, that interacts with the cashier to accept payment for the purchase of car parts and gasoline;
- A tracking subsystem, that logs all purchases and tracks the inventory remaining.

The questions in this section have to do with maintenance issues in the implementation of the system, and with the operation of the system once it reaches the maintenance phase of the lifecycle.

1. While implementing the system, the development team has given some thought to the type of maintenance changes the system will require. The first step in doing this might be to:
   a. Classify the system as an S-type system.
   b. Classify the system as a P-type system.
   c. Classify the system as an E-type system.
   d. Recognize that this system is likely to require no maintenance activities.

2. The development team also recognizes that certain attributes of the system itself may make it easier or harder to maintain. Which of the following

statements about the system are likely to affect the effort required to make changes?

   a. The GSCS must respond to customers in real time.
   b. The requirements and design are well-documented.
   c. The GSCS must interface with several different pieces of hardware, such as the cash register, the gas pumps, and the credit card systems.
   d. A and B
   e. A and C
   f. B and C
   g. A, B, and C

3. TRUE or FALSE: While implementing the system, the team tracks seven measures of software complexity, on the assumption that the most complex modules will be likely to require the most future maintenance. A reasonable way to minimize the data collection effort would be to select the one measure from this set that seems best correlated with maintenance effort and discard the rest.

4. On the past several projects, the team has tried to use a predictive model that estimates the amount of maintenance required by a system based on the code complexity measures, among other factors. However, the predictions seem to consistently underestimate the actual effort required by about 40%. Which of the following is a valid assessment of the model?
   a. It suffers from bias, which should be assessed with a u-plot.
   b. It suffers from noise, and should be assessed using the prequential likelihood function.
   c. It suffers from both bias and noise and should be discarded.

5. TRUE or FALSE: The prediction system described in question 5 is valid only if the acceptance range is greater than 40%.

6. TRUE or FALSE: If a measure (such as one of the complexity measures from question 3) were not valid for predicting effort, it could not be internally valid.

While implementing the system, the development team keeps in mind the Belady-Lehman equation of maintenance effort. They would like to use this equation as a guide that will hopefully allow them to save effort during the maintenance phase. According to this equation, are the following expectations of the development team TRUE or FALSE?

7. A system developed using good software engineering principles will be slightly easier to maintain than one that hasn't used these principles.
8. The best use of resources would be to require someone unfamiliar with the system to perform the maintenance, since that person is unlikely to make the same mistakes or assumptions as the original development team.
9. All else being equal, if the development team is equally familiar with two systems from different environments, and the systems are equally complex, the expected maintenance effort is roughly equal.

After the system is completely implemented and has been in operation for some time, a number of changes have been identified that should be made to the system.

10. As changes are made to the system, which of the following would be reasonable to expect?

   a. If enough new functionality is added, it will eventually be more cost-effective to rewrite the GSCS rather than continue modifying it.
   b. The number of modules in the code will increase and the connections among them will become more complicated.
   c. Measures of the programming process, such as productivity of the maintenance team, will vary greatly as the system changes over time.
   d. A and B
   e. A and C
   f. B and C
   g. A, B, and C

11. One of the credit card companies upgrades its system for handling credit card payments, and this requires a slight change to the type of data that the GSCS needs to send to it. This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

12. The gas station owner has stipulated that the GSCS should be able to handle additional gas pumps, if the station decides to invest in them in the future. However, the development team realizes that the way in which it handles concurrency will not scale up if more gas pumps are added at the gas station. This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

13. An additional service is added for customers at the gas station. (Customers can now rent parking spots.) This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

14. When receipts are printed, if the customer's name exceeds a certain length then the purchase price does not fit on the receipt and is not printed. This situation does not occur very frequently (at most, once a week). This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

15. The situation described in question 14 represents a problem with the quality of the system because it represents a reduction in
    a. Reliability
    b. Integrity
    c. Consistency
    d. a and b
    e. a and c

f. b and c
g. a, b, and c

16. After the problem discussed in question 14 is identified, one of the developers redesigns a small part of the design to fix the problem, and changes the code accordingly. She then updates the requirements document so that the functionality now in the system is explained correctly. This is an example of:
    a. Maintaining vertical traceability
    b. Maintaining horizontal traceability
    c. Both a and b
    d. Neither a nor b

17. Operation of the system also reveals problems with the way it handles concurrent users at different gas pumps. Upon investigation, it was discovered that this problem stems from a module that was specified correctly in the requirements and design, but was implemented incorrectly in the code. This problem might have been discovered earlier if the team had used an appropriate:
    a. Linker
    b. Debugging tool
    c. Cross-reference generator
    d. Static code analyzer

18. Suppose that we want to evaluate the quality of the GSCS using Boehm's quality model. From which of the following perspectives would we assess the utility of the system?
    a. The owner of the gas station
    b. The cashiers and customers at the gas pumps
    c. The maintainers of the system
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

19. The team members who worked on the gas purchase subsystem used a new CASE tool, and they are claiming that it should be adopted by the entire team. The team leader decides to investigate whether team performance would really be improved in this way. As a basis for his evaluation, he interviews several team members and looks for trends and patterns in their responses. He asks members who used the tool questions such as: whether the use of the tool led to more frequent or characteristic kinds of problems, whether the tool was reliable, and what kinds of tasks the tool was used for. He also talks to team members who did not use the tool, in order to see if they experienced problems that using the tool might have avoided. The team leader will then try to relate this information to any differences in productivity between team members who used the tool and those who did not. This type of investigation would best be described as a:
    a. Feature analysis
    b. Case study
    c. Survey
    d. Formal experiment

20. The type of investigation described in question 19 is probably a good choice for an initial answer to this question, because:

a. The effects of potentially confounding factors can be easily eliminated during the analysis, so that any relationship between tool use and productivity will be easy to see.
b. This type of investigation is well-suited to retrospective data, and thus good use can be made of data already collected for other purposes.
c. It ensures that the data collected about the tool will be representative of all important types of users.
d. a and b
e. a and c
f. b and c
g. a, b, and c

21. Based on the initial investigation described in question 19, the tool looks promising for use by the development team. The team leader would like to run one more small study to confirm this indication. He decides that the most appropriate type of study will be a formal experiment. He constructs a small programming assignment that he feels should take only a few hours, and gives the assignment to two groups of developers who have agreed to participate. Members of one group are asked to program the solution as they normally would, while the second group is asked to come up with a solution using the tool. The first group has no access to the tool, and cannot use it; and the team leader can examine the files produced by the tool to make sure the second group actually did use it as expected. Because there is a learning curve involved in use of the tool, the second group consists of developers who used the tool on the last project. The team leader can then study the quality of the solutions produced, and the effort required, to assess how useful the tool would be. The above study suffers from which of the following pitfalls?
    a. Bias
    b. Homogeneity
    c. Misclassification
    d. a and b
    e. a and c
    f. b and c
    g. This study has none of these pitfalls.

22. TRUE or FALSE: Assume that any pitfalls identified in question 21 are fixed. The team leader can be very confident that the results seen from the study would apply if the team used the tool on a real project.

23. TRUE or FALSE: The study described in question 21 directly tests the following hypothesis: “Using the tool produces better quality software than using the normal development method in this environment.”

24. After studying the issue carefully, the team leader is convinced that the tool would be useful to the group, and acquires it for the next project. To assess whether his decision was a good one, the team leader monitors the number of hours developers actually spend using the tool. However, it turns out that, compared with the last project, developers on this project end up using it much less. The team leader can reasonably conclude:
    a. The developers would find the tool more useful, if only they would use it more.
    b. This project is different in some way from the last one, which makes the tool less applicable.
    c. This project is simply smaller than planned, and requires less development activity.

d. a and b
e. a and c
f. b and c
g. a, b, and c

Once the project is completed and some maintenance tasks are taken care of, the team leader decides to spend some time reviewing the team's software development process, in order to identify potential improvements that can be made for the next project.

25. TRUE or FALSE: The team leader's normal post-project activity is to schedule some time for a one-on-one interview with each member of the team. He asks each member how he or she felt about the last project. He allows them to talk about organization, process, or anything else they find important and does his best not to ask leading questions or to give his own opinion. This approach is an optimal way to conduct post-mortem analyses.

26. TRUE or FALSE: The team leader decides to try something new for this project: A “Project History Day” designed to track down the root causes of problems experienced while developing the GSCS. He invites the entire development team to the full-day meeting, expecting each member to raise any important problems encountered and the entire team to participate in discussing how to avoid it in the future. This Project History Day should be expected to be a successful tool for process improvement.

27. TRUE or FALSE: The final step of the team leader's process improvement effort is to produce a report to share the team's process discoveries with managers and other developers in the organization. The team leader is careful to include positive as well as negative findings. The top three problems of the last project are discussed in detail, along with suggested ways of fixing them. This strategy is an optimal way of publishing postmortem analysis results.

28. The team has also considered CMM as a way to improve their software development process. Which of the following accurately describes the CMM?
    a. It is meant to be used by a software development organization, which can use the key process areas to determine which aspects of their development process to improve.
    b. It is meant to be used by software customers, who can use it to assess the strengths and weaknesses of the software developers with whom they contract.
    c. The highest CMM ranking corresponds to the situation in which the software development process is understood simply as a “black box” that converts the inputs to the process into quality software.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

29. TRUE or FALSE: The ability to change the software development process based on lessons learned from previous projects is achieved at an early level of the CMM ranking system, and allows an organization to progress to higher levels.

30. TRUE or FALSE: One criticism of the CMM is that it assesses only the technical quality of an organization and largely ignores business quality.

31. Which of the following statements best capture the difference between CMM and SPICE?
    a. CMM defines desirable practices which serve as a benchmark for comparison.
    b. The method for performing a SPICE evaluation is prescribed so as to be as objective as possible.
    c. SPICE addresses processes, distinguishing between base and generic practices.
    d. a and b
    e. a and c
    f. a and c
    g. a, b, and c

32. In contrast to the CMM, the people capability maturity model
    a. Is aimed at assessing the capability of the developers comprising the development organization.
    b. Focuses more on the software developers themselves as a resource of the software organization, and less on the technology used by the organization.
    c. Awards a high level of maturity to an organization that has a quantitative understanding of how its practices are increasing the critical skills of its staff.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

After the GSCS has been deployed and begun stable operation, the development team starts on their next project, building an inventory tracking system for a convenience store.

33. One of the first things the development team realizes is that the inventory system from the GSCS can be largely reused in the new system, with minor modifications. If they are able to reuse modified parts of the GSCS in the new system, this type of reuse could be described as:
    a. Producer, black-box reuse
    b. Producer, white-box reuse
    c. Consumer, black-box reuse
    d. Consumer, white-box reuse

34. Because of situations like the one described in the previous question, the development teams decides that they would like to look into creating a reuse library. Which of the following are reasonable expectations about investing in reuse?
    a. It will improve the speed at which prototypes can be constructed.
    b. It will produce concrete benefits in the short term.
    c. It will help reduce time spent on unit testing as well as on coding.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

35. TRUE or FALSE: Studies have shown that, under certain conditions, code reuse can actually lead to more fault-prone software than writing code from scratch.

36. TRUE or FALSE: To help decide whether or not investing in a reuse library would be worthwhile, the team looks for “testimonial evidence.” This means they would like to base the decision on information from sources such as direct observation of code libraries in use and second-hand information from other developers who have experience with them.

37. TRUE or FALSE: The credibility of the evidence described in the previous question is related primarily to its accuracy.

38. If the team lead analyzed the team as being in the “mainstream market” in terms of adopting new technologies (as opposed to being in the class of “innovators” or “early adopters”), which of the following would be true?
    a. The team would be willing to invest some of its own time into figuring out for themselves how to install and use the library, if support was not available.
    b. The team would be willing to invest in a reuse library only after there was widespread evidence of its effectiveness.
    c. An important factor in the decision would be what other members of the organization’s business community are doing.
    d. A and B
    e. A and C
    f. B and C
    g. A, B, and C

Review Exam 4 Answers

1. b; The GSCS is a P-type system, since the problem can be described directly and completely, and has an exact solution. Unlike an E-type system, the system is not embedded in the environment, that is, the practical abstraction of the problem is unlikely to change due to an improved understanding resulting from the solution. As a P-type system, incremental change is possible in order to improve the solution. [Section 11.1]
2. g; In general, a system with real-time requirements is more difficult to change than a system without such requirements. The existence of documentation affects maintenance effort since a system without well-documented code and design can be almost impossible to search through for a problem's solution. The need to interface with different types of hardware can also be expected to affect maintenance effort since it is possible that the code will require changes every time any piece of hardware is upgraded or replaced. [Section 11.3]
3. FALSE; Section 11.4 of the textbook describes how it is necessary to be cautious in using only one measure to represent complexity, since each measure captures only one attribute of the system, and there are many attributes that contribute to complexity.
4. a; The prediction is biased because it is consistently less than the actual value. However, because the predictions do not fluctuate as underestimates and overestimates of the actual value, we cannot say the prediction is noisy. [Section 13.1]
5. TRUE; A prediction system is said to be valid if it makes accurate predictions. The acceptance range specifies the accuracy required of the model. [Section 12.3]
6. FALSE; The internal validity of a measure (how well it measures the attribute it claims to) should not be rejected just because it is not a part of a valid predictive system. The textbook uses the example of lines of code, which is a valid measure of program size but which is not always useful as a predictor of faults. [Section 12.3]
7. FALSE; The variable 'c' is reduced by software engineering practices and is an exponential term in the equation. Thus even small changes to 'c' have the potential to greatly affect the expected effort. [Section 11.3]
8. FALSE; The variable 'd', the team's familiarity with the software, decreases the expected effort as it decreases. [Section 11.3]
9. FALSE; The relevant variable here is 'K,' which is an empirical constant that depends on the environment. [Section 11.3]
10. d; The fourth "Law of Software Evolution" states that there will be no significant fluctuations in organizational attributes, such as productivity. [Section 11.1]
11. b; This change is necessary for the system to adapt as it evolves over time. [Section 11.2]
12. d; This change involves modifying part of the system to prevent future faults. [Section 11.2]
13. b; This change is necessary for the system to adapt as its requirements evolve over time. [Section 11.2]
14. a; This change is necessary to directly correct a fault. [Section 11.2]
15. d; Since a given input will always result in the same behavior there is no reduction in consistency. (Whether the failure occurs or not depends entirely on name length.) However, this failure means that the system does not always produce the desired result (the customer may not receive a useful receipt of his or her purchase), which represents a reduction in integrity. The existence of this problem also reduces the amount of time

the system can run between failures, representing a reduction in reliability. [Section 12.4]

16. b; Horizontal traceability addresses the relationships within a collection of work products, such as requirements, design, and code documents which all describe the same system. Vertical traceability, in contrast, addresses relationships between the subcomponents of one such work product. [Section 11.5]
17. c; Cross-reference generators assist developers by representing the traceability between various system work documents. If the team had used a cross-reference generator, it would have been more likely to note that components of the requirements and design were not represented in the code. [Section 11.5]
18. g; Boehm's model assesses utility from the viewpoint of three types of users. First, the customer of the system (the gas station owner) who is pleased with the utility if the system performs according to his or her requirements. Secondly, the users of the system (cashiers and gas station customers) who will be assumed to be pleased with the utility if the system operation will not change as components are upgraded or replaced. Thirdly, the maintainers of the system who will be pleased if the system is easy to understand and make changes to, as necessary. [Section 12.4]
19. c; A survey is a retrospective study that attempts to discover the effects of a method, tool, or technique on participants. [Section 12.1]
20. b; Surveys are well-suited to the use of retrospective data. However, because the experimenter cannot manipulate key variables in the environment, it is difficult to assess the impact of potentially confounding factors. For example, the experimenter cannot ensure that the data collected will be representative of different types of users, since the experimenter cannot determine the makeup of the population sampled. [Section 12.1]
21. a; This study suffers from bias because the assignment of developers to groups (those who use the tool versus those who don't) may affect the results. Because all of the developers who are in the "tool use" group have previously adopted the tool on their own, we might wonder whether the tool is somehow better suited to their preferred working style. That is, the results seen for this group may not be the same as those seen for a random group of developers. The study does not suffer from homogeneity or misclassification because the two groups will have different levels of the factor (tool use) and the experimenter can check whether or not the groups performed as instructed. [Section 12.2]
22. FALSE; There is an additional pitfall, namely that the study is too short to expect its results to scale up directly to a real project. It is possible that the study would underestimate the usefulness of the tool, since there may be a learning curve that cannot be overcome in the few hours allotted to the study. Conversely, the study may overestimate the usefulness, since the tool may work well for very simple problems but not apply well to something as complicated as a real project. [Section 12.2]
23. FALSE; The hypothesis given cannot be tested directly because "software quality" is not quantifiable. That is, the experimenter will have to select a way of measuring software quality that can then be tested in the experiment. [Section 12.2]
24. g; Section 13.2 discuss an analogous situation in which inspection effectiveness is studied. In interpreting results of this sort, it is always necessary to keep in mind alternative explanations for observations: for example, human subjects don't always use technologies in the way researchers expect them to, and there are usually large variations between projects.

25. FALSE; Ideally, the team leader would attempt to solicit this information while preserving the anonymity of the respondents, in order to be more certain that the information is not biased. Also, the unstructured nature of the interviews does not allow comparison across projects, because there is no guarantee that the team members will focus on the same issues at each interview. [Section 12.5]
26. FALSE; A better approach would be to identify issues for discussion beforehand. This approach would help keep the discussion focused, and could help make sure only people with relevant experience are invited to participate in the discussion. [Section 12.5]
27. TRUE; This description matches the guidelines given in the textbook for successfully publishing the results. In particular, while much benefit can be gained from distributing such a report to peer developers, managers are also stakeholders in the software development process and can benefit from insight into the process. [Section 12.5]
28. d; The CMM identifies important "key process areas" against which organizations can be assessed by themselves and customers. At the highest CMM level, the organization is expected to have achieved greater insight into what goes on within its software development process; it should not be considered a "black box." [Section 12.5]
29. FALSE; The ability to change the software process based on previous lessons learned is achieved at the highest level of the CMM. [Section 12.5]
30. TRUE; The CMM focuses on improving detection of faults, time to market, and operational failures, but ignores such business quality measures as customer satisfaction or appropriateness of functionality. [Section 13.3]
31. c; In contrast to the CMM, which evaluates organizations, SPICE evaluates processes. Both CMM and SPICE define a set of "best" practices for comparison and attempt to perform the comparison as objectively as possible. [Section 12.5]
32. g; The people capability maturity model is focused on improving the skills of individual developers and teams, and does the by helping the organization build up a quantitative understanding of how it can contribute to improving the critical skills of its staff. This quantitative understanding occurs at the second-highest level of maturity, the "managed" level. The key process areas are concerned with the skills of individuals and teams, not the technology they use. [Section 12.6]
33. d; Since this situation deals with the use, not the construction, of reusable components it represents consumer reuse. Since the subsystem will have to be modified before being reused, this will be white-box reuse. [Section 12.4]
34. e; Because early prototypes may be constructed using some of the reusable functionality, these prototypes could be constructed more quickly with a reuse library. (Later versions of the system will presumably require the reusable components to be tailored more, so the time savings will not be as great.) Also, if components are unit tested before being placed in the library and then reused verbatim, they need not be tested again. However, creating a useful reuse library requires a large initial effort (necessary for identifying reusable components, making them as reusable as possible, and then testing them thoroughly) and so may not produce large concrete benefits in the short term. [Section 12.4]
35. TRUE; The Moeller and Paulish study described in section 13.2 shows that, in a particular environment, reused components that required some changes (about 25% of the lines of code) were more fault-prone than components written from scratch.
36. TRUE; “Testimonial evidence” includes direct observation, second-hand experience, and opinion. [Section 14.2]

37. FALSE; Credibility of evidence is also related to the credibility of the source. [Section 14.2]
38. F; Mainstream market adopters are generally only willing to invest in a new technology when widespread evidence of its effectiveness and adequate support and training are already available. [Section 14.2]

Final Exam

The following questions are in reference to a hypothetical “Loan Arranger” system. The Loan Arranger is meant to assist a “consolidating organization”, a specific type of business in the financial domain.

Bank customers often borrow money from banks; they promise to repay the loan plus interest over a certain period of time. Each of these loans is an ultimately lucrative proposition for the bank, but usually a long period of time is required for the bank to collect its earnings. Also, there is always the chance that a borrower will be unable to repay the loan. Consolidating organizations make money by buying loans from banks and reselling them to investors. Loan analysts, employed by the consolidating organization, work with investors to select a set of loans that meet their requirements in terms of time, risk, and initial purchase price.

The Loan Arranger application has two main types of functionality. First, it tracks all of the loans that are owned by the consolidating organization. (The entire set of loans owned by the organization is known as its “portfolio”.) In order to track loans correctly, the Loan Arranger must interface with the banks to find out about new loans that should be added to the portfolio, or updates to loans that are already in the portfolio. Secondly, the Loan Arranger allows a loan analyst to select a subset of loans from the portfolio that match an investor's desired investment characteristics (these subsets of loans are referred to as “bundles”). The loan analyst can either select the bundle manually, or ask the system to select an optimal bundle given the investor's constraints. The Loan Arranger then updates the portfolio when the bundle is purchased by the investor. There are numerous constraints on how this functionality is to be achieved; most importantly, the Loan Arranger must be usable by more than one analyst at a time.

As a first step in developing the Loan Arranger, the development team meets with a representative from the consolidating organization to formulate a set of requirements.

1. Which of the following statements best describe the benefits that the consolidating organization may expect?
    a. The requirements process can help the consolidating organization think more clearly about the performance issues that are necessary for the Loan Arranger (for example, the maximum amount of time allowable for a search through the portfolio).
    b. The requirements process can help the development team communicate with the representative from the consolidating organization about the system.
    c. The requirements process can help the development team specify the types of data structures that will be used in the Loan Arranger, in order to ensure that the performance requirements are met.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

2. The development team needs to pick a representation for the requirements. Which of the following are valid choices and rationales?

    a. Structured Analysis and Design Technique, because it may be useful to view the system at different levels of detail at different times (for example, it may be useful to specify the inputs and outputs of the system before proceeding to design the mechanisms by which loans are included in bundles to be sold).
    b. Warnier diagrams, because they will be helpful in understanding the different types of loans that must be handled by the system.
    c. Data flow diagrams, since this will help the development team understand how the functionality can be used by multiple users simultaneously.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

Mark items 3 through 6 TRUE if they belong in the requirements for the Loan Arranger, and FALSE if they do not.

3. A description of the skills and knowledge the development team assumes that the loan analysts already have.
4. A description of how the consolidating organization decides which loans to buy from banks for inclusion in the portfolio.
5. Constraints on the maximum allowable time for the system to automatically suggest an “optimal” bundle of loans for sale to investors.
6. The hardware on which the Loan Arranger will be designed to run.

A requirements review is undertaken to make sure that the requirements adequately describe the system to be built.

7. Which of the following excerpts should be considered valid functional requirements during the review?
    a. “Once the Loan Arranger has automatically generated a bundle, the loan analyst must be able to modify it manually. The loan analyst may modify a bundle by either removing loans which are already included, or adding additional loans.”
    b. “A record should be kept for each bank from which loans are purchased, consisting of the name of the bank, the name of a contact person at the bank, and the phone number of the contact person. The loan analyst should be able to edit fields that are changeable.”
    c. “The expected profit of a fixed-rate loan is the amount of interest that will be received over the remaining life of the loan. The formula for computing loan interest is included in Appendix A.”
    d. a and b only
    e. a and c only
    f. b and c only
    g. a, b, and c

8. Which of the following can be considered examples of valid nonfunctional requirements?
    a. “If updates are made to any displayed information, the information is refreshed within five seconds.”
    b. “The application must ensure that users are limited to authorized loan analysts.”
    c. “The application must be available for use by a loan analyst during 97% of the business day.”
    d. a and b only
    e. a and c only
    f. b and c only

g. a, b, and c

In questions 9 through 12, review the given excerpt from the requirements and decide whether it is an adequate requirement or not. If the excerpt is adequate, mark choice 'e'. If it should be rewritten, mark all the reasons that apply.

9. ”A borrower can be in one of three states: 'good', 'late', or 'default'. A borrower is considered to be in 'good' standing if all loans to that borrower are in good standing. A borrower is considered to be in 'default' standing if any of the loans to that borrower have default standing. A borrower is said to be in 'late' standing if any of the loans to that borrower have late standing.”
   a. This requirement should be rewritten; it is incorrect.
   b. This requirement should be rewritten; it is ambiguous or inconsistent.
   c. This requirement should be rewritten; it is unrealistic.
   d. This requirement should be rewritten; it is unverifiable.
   e. This requirement is fine.

10. “The format of the reports is at the discretion of the individual banks. The Loan Arranger must be easily extensible, so that it can handle new file formats as necessary.”
   a. This requirement should be rewritten; it is incorrect.
   b. This requirement should be rewritten; it is ambiguous or inconsistent.
   c. This requirement should be rewritten; it is unrealistic.
   d. This requirement should be rewritten; it is unverifiable.
   e. This requirement is fine.

11. “Each loan must have a loan amount of at least $1000 but not more than $500,000. There are two types of loans: regular and jumbo. A regular loan is for any amount less than or equal to $275,000. A jumbo loan is for any amount over $275,000.”
   a. This requirement should be rewritten; it is incorrect.
   b. This requirement should be rewritten; it is ambiguous or inconsistent.
   c. This requirement should be rewritten; it is unrealistic.
   d. This requirement should be rewritten; it is unverifiable.
   e. This requirement is fine.

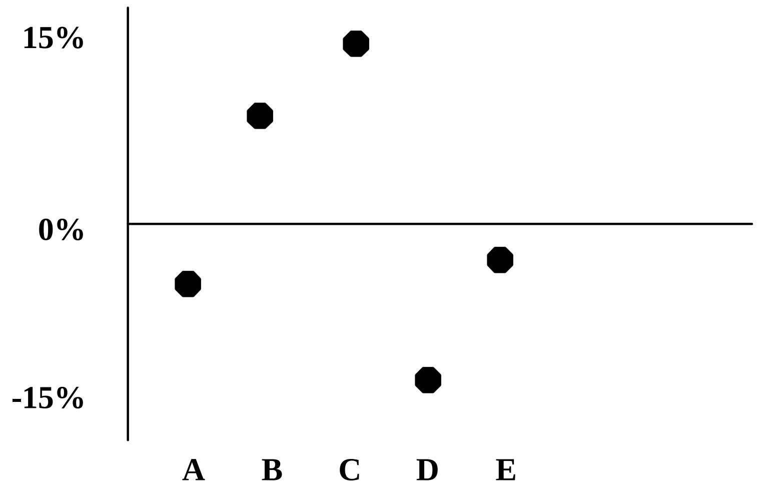
12. ”The user must be advised when a search request is inappropriate or illegal.”
   a. This requirement should be rewritten; it is incorrect.
   b. This requirement should be rewritten; it is ambiguous or inconsistent.
   c. This requirement should be rewritten; it is unrealistic.
   d. This requirement should be rewritten; it is unverifiable.
   e. This requirement is fine.

After the initial requirements have been defined, the team decides to develop an initial project plan. First, the risks are identified and assessed. Determine whether or not each of the statements 13 through 15 is describing a risk. Answer TRUE if the statement describes a risk, FALSE otherwise.

13. The developers of the Loan Arranger application do not have much experience in the financial domain.

14. The Loan Arranger application must interface with many external systems.
15. Prototyping is used on the Loan Arranger application to make sure that the team is implementing the correct functionality.

In addition to the risk management activities, cost, schedule and effort estimations are done. On the past several projects, the team has tried to use a predictive model to estimate the expected size of the project. The relative errors of the predictions on the last five projects are shown below. A relative error of 0 means that there was no difference between the predicted and actual values. A positive relative error means that the prediction was an overestimate, while a negative value indicates an underestimate.}



16. As can be seen from the graph, predictions are usually within 15% (plus or minus) of the actual effort. However, for a given prediction, it is impossible to say whether it is an underestimate or an overestimate of the actual value. Which of the following is a valid assessment of the model?
    a. It suffers from bias, which should be assessed with a u-plot.
    b. It suffers from noise, and should be assessed using the prequential likelihood function.
    c. It suffers from both bias and noise and should be discarded.

Suppose the COCOMO 2.0 estimation process has been chosen to estimate size and effort for part of the project.

17. The loan search and selection subsystem of the Loan Arranger application has 10 screens and 20 reports. Two screens are rated as difficult, 5 as medium and 3 as easy. Ten reports are rated as difficult, 6 as medium and 4 as easy. Using the COCOMO 2.0 stage 1 model, how many new object points does the subsystem have? Assume no 3GL components or reuse of existing components.
    a. 210
    b. 137
    c. 109
    d. 30
    e. None of the above.

As part of an on-going investigation into the benefits of a new size estimation technique, the size of part of the Loan Arranger application will be estimated using the experimental technique. The technique has been used on several projects in the past. The estimates generated by the technique and the actual values for project size are shown below. The criteria for a good estimating technique are: 75% of the estimates should be within 25% of the actual value; and the mean magnitude of the relative estimated errors should be less than 25%. Use the table of project size estimates and the criteria given to answer the questions about the estimation technique.

Project	Estimate	Actual
A	5000	10000
B	25000	20000
C	50000	80000
D	70000	110000
E	40000	45000
F	45000	50000

18. Given the table of estimates and actuals, what is the MMRE? Round to the nearest 1/100.
    a. 0.04
    b. 0.14
    c. 0.28
    d. 0.34
    e. 0.43

19. What is the PRED(.25)?
    a. 0.25
    b. 0.28
    c. 0.33
    d. 0.50
    e. 0.75

20. Based on the criteria for a good estimation technique and the estimate data gathered so far, can this technique be used to create good estimates for the Loan Arranger project? (Yes/No)

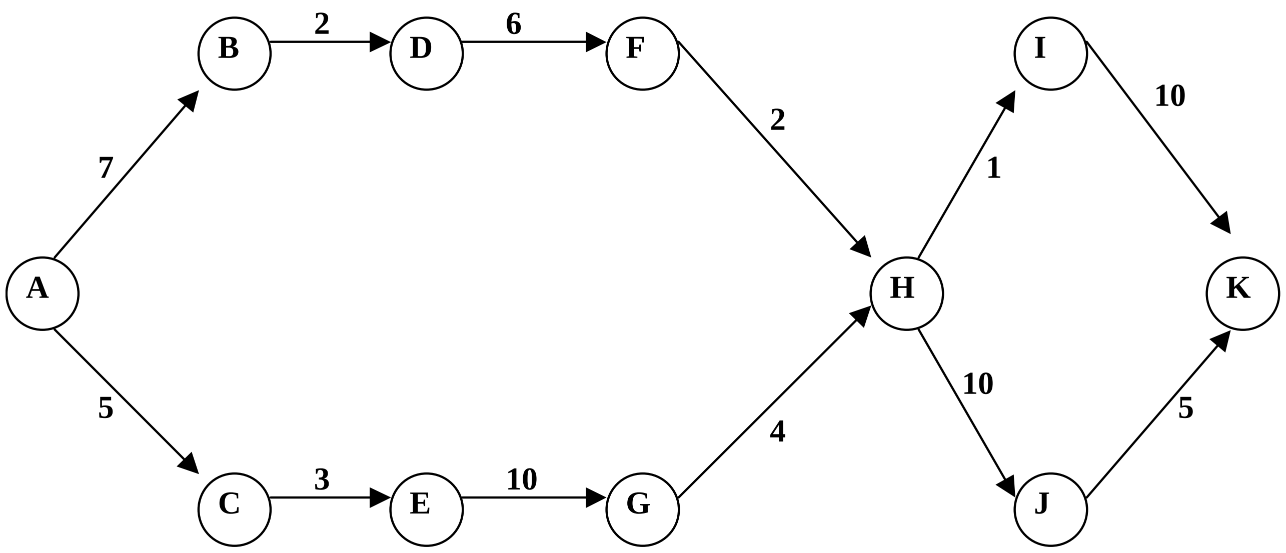
21. Several estimators use various techniques to estimate the amount of effort required for the Loan Arranger project. Each estimator arrives at his or her estimate independently. The independent estimates are 600 person-months, 650 person-months, 800 person-months and 750 person months. Using these estimates, what is the Delphi estimate for this project? Round to the closest month.
    a. 700 person-months
    b. 684 person-months
    c. 648 person-months
    d. 600 person-months
    e. 523 person-months

22. TRUE or FALSE: When the Delphi method is used and participants are allowed to iterate through the process several times, group dynamics drives the group consensus to converge close to the median of the estimates.

23. If the Delphi estimate for effort is used and there are 20 team members working on the project, how many months will the project take? Assume all team members can work concurrently. Round to the closest month.
    a. 26
    b. 30
    c. 32
    d. 34
    e. 35

24. If there are 20 team members assigned to the project team, how many potential lines of communication exist?
    a. 20
    b. 45
    c. 190
    d. 400

To build a schedule, the critical path method is used. The activity graph (shown below) is used to depict the dependencies among the activities and milestones of the Loan Arranger project. The nodes of the graph represent the milestones of the project. The edges linking the nodes represent the activities. The numbers adjacent to the edges represent the number of days required for the activity. For example, it will take 5 days to complete the activity starting at milestone A and ending in milestone C. Use this activity graph to answer the following questions:



25. Which of the following is a critical path from milestone A to milestone K?
    a. ACEGHJK
    b. ACEGHIK
    c. ABDFHJK
    d. ABDFHIK

26. What is the slack time for the activity starting at milestone F?
    a. 4
    b. 5
    c. 16
    d. 21

27. What is the length of the critical path identified in question 25?
    a. 28
    b. 32
    c. 33
    d. 37

28. Which milestones are precursors to J?
    a. D
    b. E
    c. I
    d. D and E
    e. D and I
    f. All of the above

The development team decides that the next step is to create a conceptual and then a technical design.

29. The conceptual design is felt to be of value because it will allow the consolidating organization to check:
    a. How its policy of allowing banks to specify the format of their reports will affect the system.
    b. What functions the loan analysts can access at any given time.
    c. Whether the reports generated by the system will match the report format already used by the organization.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

30. Which of the following are valid rationales for creating a separate technical design?
    a. The conceptual design will be useful for communicating with the representative of the consolidating organization; it may contain financial jargon but not implementation details or programming jargon.
    b. The conceptual design will not include detailed financial formulas so it will be independent of the implementation.
    c. The technical design should contain more detail about what data structures will need to be used in order to meet the performance requirements.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

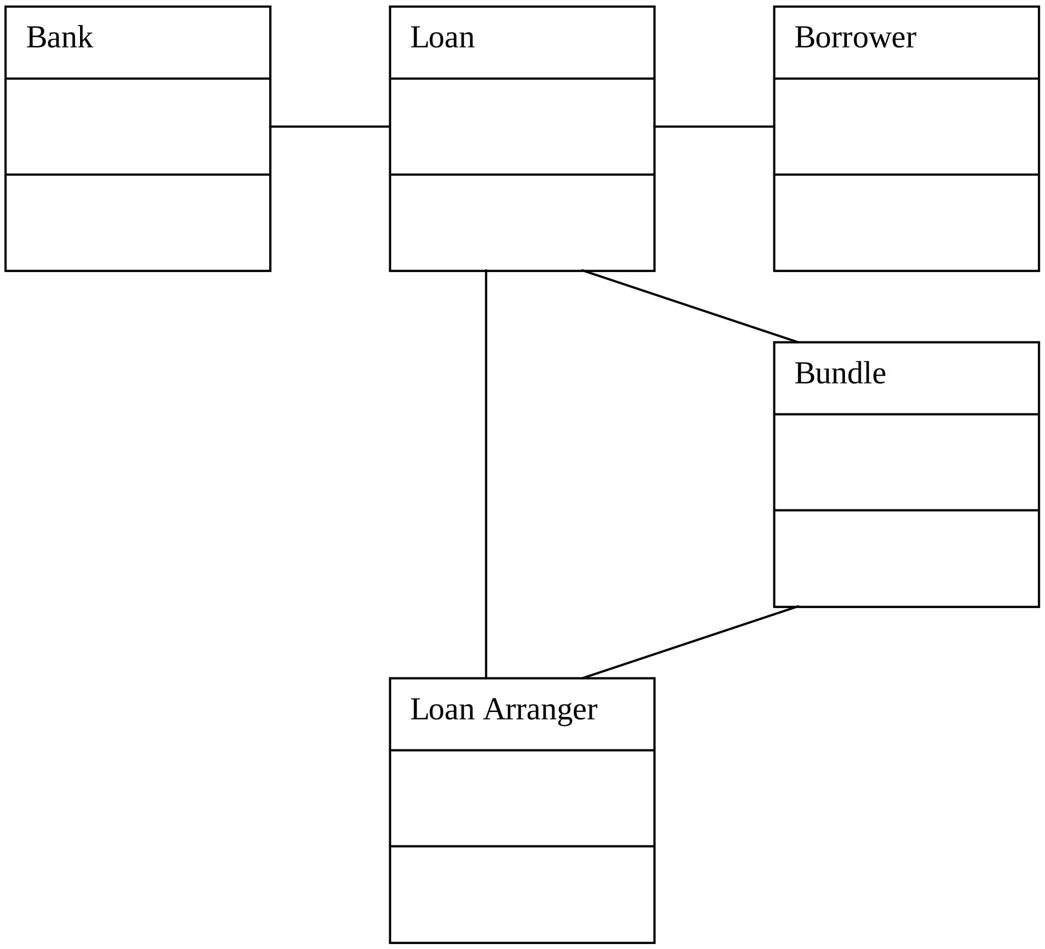
31. The team has to decide on a general approach to creating the design. Which are NOT valid choices and rationales?
    a. Data-oriented decomposition, because the data structures are central to the design of the system, and many aspects of the data structures are highly constrained by the requirements.
    b. Modular decomposition, because the system functions are highly interdependent, so the system is not easily divisible into separate subsystems.
    c. Event-oriented decomposition, because the type of functionality available to the loan analyst at any given time will depend on the

current state of the system, as determined by the actions of all loan analysts using the system.
   d. a and b
   e. a and c
   f. b and c
   g. a, b, and c

32. The team leader decides that the logical next step is to decide on an architectural style for the system. Which of the following are valid choices and rationales?
   a. Object-Oriented, since this design style would allow certain features to be updated (for example, the data structures holding the bank information) while requiring no changes to other components in the system.
   b. Client-Server, since this design style would be a convenient way to allow many different loan analysts to view the same data in different ways.
   c. Repository, since it will be convenient to view the system as a central data store (the portfolio of loans) with mechanisms for storing, retrieving, and updating the data.
   d. a and b
   e. a and c
   f. b and c
   g. a, b, and c

33. Since the development team has some experience with concurrency from a previous project, the technology involved in this system is well understood. The team has decided on an Object-Oriented design which breaks the system into several components, and the team members are fairly confident they understand how each of the components will achieve the required functionality. On the other hand, it is not yet clear how the components will interact with each other and with external systems. A reasonable strategy for completing this design would therefore be:
   a. Prototyping

   b. Fault-tree analysis
   c. Design by contract



This is the class diagram for the initial high-level design of the Loan Arranger system.

34. Which of the following best describes the amount of coupling for class Loan, relative to the other classes in the high-level design shown above?
    a. High coupling
    b. Low coupling
    c. Can't tell

35. Which of the following best describes the amount of cohesion for class Loan, relative to the other classes in the high-level design shown above?
    a. High cohesion
    b. Low cohesion
    c. Can't tell

36. Based on what is known about the level of cohesion and coupling for class Loan, it is reasonable to assume that:
    a. It will probably be easier to modify than class Borrower.
    b. It will probably be harder to modify than class Borrower.

   c. No conclusions can be drawn about the ease of modification.

37. After an internal review, some of the designers want to add more classes to the high-level model. Which of the following are appropriate for a high-level, conceptual design?
   a. Classes “Variable Rate Loan” and “Fixed Rate Loan,” subclasses of “Loan” which are handled differently by the consolidating organization.
   b. Class “Acceptance Dialog,” which controls the window that appears on the screen when a new loan can be purchased by the consolidating organization.
   c. Class “Identification Number,” which consolidates the implementation for unique loan ID numbers in the system.
   d. A and B
   e. A and C
   f. B and C
   g. A, B, and C

38. To estimate the amount of effort that will be needed to construct the system, the team could look at which of the following metrics during system design?
   a. Lack of cohesion of methods
   b. Number of key classes
   c. Number of support classes
   d. A and B
   e. A and C
   f. B and C
   g. A, B, and C

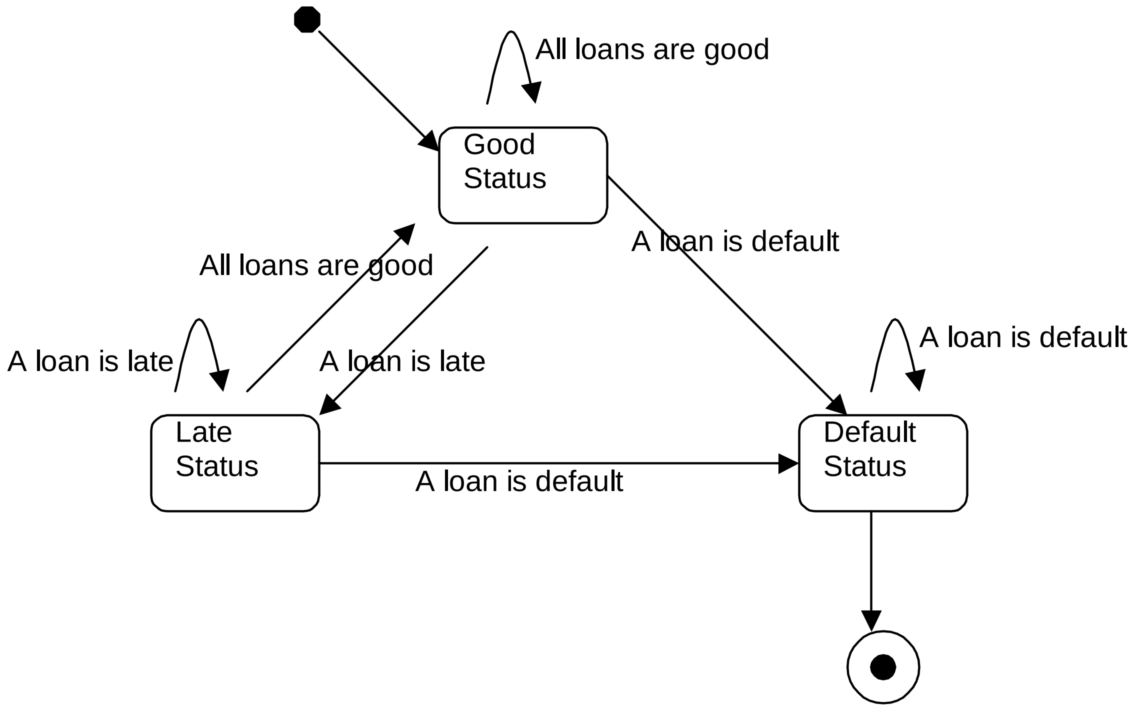
39. TRUE or FALSE: The Weighted Methods per Class (WMC) metric is useful for identifying classes that are potentially reusable classes. These classes are likely to be the ones with the highest values of the metric.

40. TRUE or FALSE: Number of Children (NOC) is useful for identifying those classes on which greater testing effort should be spent. These are the classes with high values for the metric.

41. TRUE or FALSE: Response for a Class (RFC) is useful for identifying those classes on which greater testing effort should be spent. These are the classes with high values for the metric.

42. The following figure represents the state diagram created during design for class “Borrower.” Which of the following statements are true?
   a. Regardless of the state of the borrower, if any loan is default, the borrower is considered in default status.
   b. Regardless of the state of the borrower, if any loan is late, the borrower is considered in late status.
   c. Regardless of the state of the borrower, if any loan is good, the borrower is considered in good status.
   d. A and B.
   e. A and C.
   f. B and C.
   g. None of these.

43.	TRUE or FALSE: It cannot be determined what state an object of type Borrower will be in when it is created.



As a next step, the high-level design is expanded into a low-level design.

44. In the low-level design, the portfolio is represented as a central data store. This data store is accessed (and potentially modified) by a number of other system components. For example, one component updates the data store when new information is received from a bank, while another component searches the data store for loans to create a bundle. The relationship between these components and the data store is best described as:
    a. Content coupled
    b. Common coupled
    c. Control coupled
    d. Stamp coupled
    e. Data coupled

45. Which of the following items of information are NOT appropriate for inclusion in the final low-level design?
    a. The format of the screens that will be used by the loan analyst
    b. The format of the reports the system will generate for use by the loan analyst
    c. The format and storage specifications of archival reports
    d. None of the above (all are appropriate)
    e. a and b
    f. a and c
    g. b and c
    h. a, b, and c

From previous projects, the team leader has realized that the team typically experiences many problems in the implementation phase due to poor decisions made in the design phase. To address this problem, the team leader decides to make a process change: the addition of critical design reviews to the team's software development process.

46. In preparing for the critical design review, which of the following would be appropriate actions for the team leader?
    a. Invite program designers to the reviews, instructing them to gain a better understanding of the design.
    b. Invite program designers to the reviews, instructing them to critique the existing design.
    c. Require the design to be redone and schedule added reviews, if major problems are discovered during the review.
    d. a and b
    e. a and c
    f. b and c
    g. a, b, and c

47. The critical design reviews discussed in question 46 require a lot of extra time from the developers: for planning, preparing for, and then actually holding the meetings. The team leader would like some empirical indication whether this additional investment in design time actually pays off. His idea is to introduce critical design reviews on the Loan Arranger project only, and to compare the results on this project to the results on the development team's previous projects. As much as possible, he intends to make sure that everything else about the Loan Arranger project is typical of the kinds of projects the team usually works on. If key factors on the Loan Arranger are not typical for other projects of this team, then he will at least document those key factors and reason about their possible influence on the results. This type of study would be best described as a:
    a. Feature analysis
    b. Case study, with sister projects
    c. Case study, with baseline
    d. Case study, with random selection
    e. Survey
    f. Formal experiment

48. A benefit of the type of study identified in question 47 is that:
    a. Any differences in the implementation phase can be directly attributed to the use of critical design reviews.
    b. The more typical both the Loan Arranger project and the comparison projects are, the more confidence there is that differences in the implementation phase are due to the use of critical design reviews.
    c. No conclusions can be drawn about the effects of critical design reviews, but it can be determined how software developers on this team will react to the new process.

Once the design of the system has passed the critical design review, the project moves into the implementation phase.

49. To best implement the system, the development team has to give some thought to the type of maintenance changes the Loan Arranger system will eventually require. Since few (if any) changes are expected in the way the consolidating organization tracks and bundles loans, the team should:
    a. Classify the system as an S-type system.
    b. Classify the system as a P-type system.
    c. Classify the system as an E-type system.

   d. Recognize that this system is likely to require no maintenance activities.

During implementation, several problems occur. Identify the problems as errors, faults or failures.

50. A developer implementing a component to keep track of loan updates thinks that the only modifications to loans will be additions and deletions of loans. The developer doesn't realize that interest rates on loans may change also.
51. A loan analyst notices that the loan total (original loan amount + interest) displayed on the screen is incorrect. The total value is less than the original loan amount.
52. In the code for computing loan profit, the initial purchase price is added to the profit. It should be subtracted.
53. When a loan analyst conducts three optimal bundle searches using different search criteria each time, the same set of loans is always returned. Not all of the loans returned match the criteria specified by the loan analyst.

While implementing the Loan Arranger, the development team finds out it is likely that they will be doing additional projects in the financial domain.

54. As part of the Loan Arranger, the team implements a module that calculates the interest that will be paid on a fixed-rate loan. Because it is likely that any future projects in the financial domain will also need to use this function, the module is designed to be reusable, with clearly defined and simple inputs and outputs. It is intended that any future projects can thus reuse the module directly, as long as they know the correct inputs to send and the correct output format to expect. This is an example of:
   a. Producer, black-box reuse
   b. Producer, white-box reuse
   c. Consumer, black-box reuse
   d. Consumer, white-box reuse

55. Suppose that a future project does in fact have need of a module to calculate fixed-rate interest. Which of the following problems may stand in the way of effectively reusing the module described in question 54?
   a. The developers of the future project will have to search through all of the components available for reuse, and may not find the module.
   b. The developers of the future project may not be properly trained, and may not even recognize the situation as a potential for reuse.
   c. The developers of the future project may not be motivated to reuse, and may end up re-implementing the functionality from scratch.
   d. a and b
   e. a and c
   f. b and c
   g. a, b, and c

When implementation of a module is complete, it undergoes code review. In the following program fragments from the Loan Arranger application, identify violations (if any) of good programming style that should be caught during review.

Use the following choices in your response:}
(a) Generality
(b) Efficiency

(c) Formatting
(d) Documentation
(e) No violations

56.
```
void PrintLoanList(LoanList *loans){
LoanList *l = loans;

while (l){
cout << *l << "\n";
l = loans->next;
}
}
```

57.
```
float ValidateLoans (LoanList &bundle, LoanList &loans){
/* Validate and calculate the total profit of the loans in the bundle.   */
/* If a loan in the bundle  does not exist in the loans list,            */
/*    return -1.                                                         */
/* If all loans in the bundle exist in the loans list,                   */
/*    return the total profit for all loans in the bundle.               */

float total = 0;

for (int i=0; i < bundle.getcount(); i++)
if (!loans.Exists(bundle[i])) return -1;
for (int j=0; j < bundle.getcount(); j++)
total += bundle[i].getprofit();
return total;
}
```

58.
```
/* if the loan amount is greater than 275K, it is a jumbo loan */
if (l->getamount() > 275){
type = JUMBO;
if (l->getprofit() > 10)
/* if the profit is greater than 10K, add the loan to the bundle */
bundle->addloan(l);
}
else /* a regular loan */
type = REGULAR;
```

During code reviews of the Loan Arranger, the following faults were identified. Classify the type of fault in each code fragment.

59.
```
float ComputeProfit(float initial, float rate){
float profit;

profit = (1+rate)*initial + initial;
return profit;
}
```

In this fragment, the function used to calculate the profit is incorrect.

a. initialization fault
b. computation fault

c. precision fault
d. b and c only
e. a and b only
f. none of the above

60. LoanList::~LoanList(){

```
/* delete all of the elements in the list */
for (int i=1; i < count; i++)
delete list[i];
}
```

In this fragment, the first item of the list (list[0]) is not deleted.

a. initialization fault
b. computation fault
c. precision fault
d. b and c only
e. a and b only
f. none of the above

61. Loan *loans[10];

```
for (int i=0; i<=10; i++) loans[i]->setprofit(0);
```

In this fragment, the loop includes an operation on loans[10] which is not part of the array.

a. initialization fault
b. precision fault
c. capacity or overload fault
d. a, b, and c
e. none of the above

62. One module of the Loan Arranger describes its functionality by means of the following assertions:

A_1: (T is an array) & (T is of size N) & (S = 0)

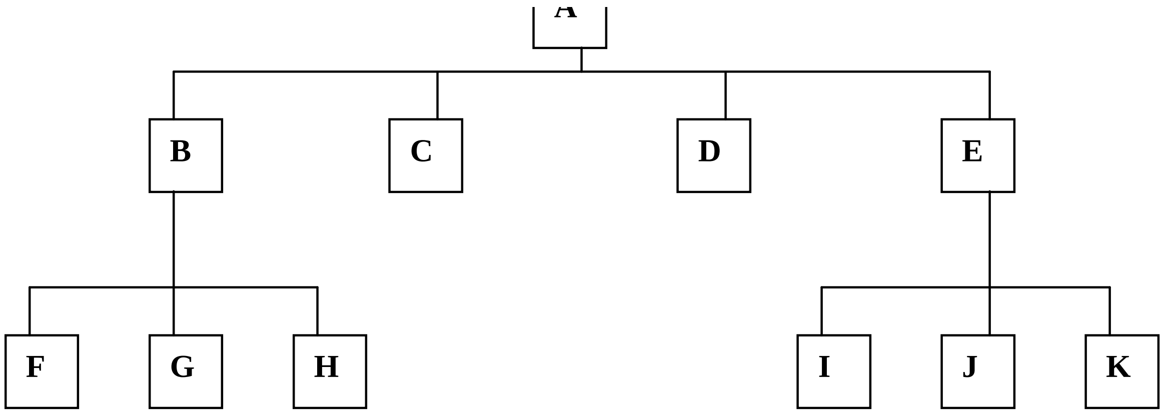
A_{end}: (T' is an array) & (T' is of size N) & $S' = \sum_{i=1}^{N} T(i)$

Choose the statement that best describes what is happening between the two assertions.

a. The values of array *T* are being assigned to the array *S*'.
b. The values of array *S* are being added to the values of array *T*'.
c. The values of array *T* are being added to the values of array *S*'.
d. The sum of the values of array *T* are being assigned to *S*'.
e. None of the above.

When implementation of the Loan Arranger is nearly complete, testing begins.

The figure below shows the component hierarchy of the Loan Arranger application. Use this figure to identify the testing strategy indicated by the sequences given. The “;” is used between test sets and each test set is represented as a comma-separated list. For example, the sequence {F,G};{B,F,G} means that components F and G were tested first. Then, components B, F and G were tested.



63. {A};{A,B,C,D,E};{A,B,C,D,E,F,G,H,I,J,K}
    a. Top-down testing
    b. Bottom-up testing
    c. Sandwich testing
    d. Big-bang testing
    e. Modified top-down testing

64. {F};{G};{H};{I};{J};{K};{B,F,G,H};{C};{D};{E,I,J,K};{A,B,C,D,E,F,G,H,I,J,K}
    a. Top-down testing
    b. Bottom-up testing
    c. Sandwich testing
    d. Big-bang testing
    e. Modified top-down testing

65. {A};{F};{G};{H};{I};{J};{K};{B,F,G,H};{C};{D};{E,I,J,K};{A,B,C,D,E,F,G,H,I,J,K}
    a. Top-down testing
    b. Bottom-up testing
    c. Sandwich testing
    d. Big-bang testing
    e. Modified top-down testing

The following issues were caught during testing. For questions 66 through 69, identify the type of testing most likely to have discovered the defect.

66. In the component that searches for the optimal loan bundle, the investor's desired loan characteristics are not initialized properly. The parameter containing these characteristics is ignored. Which type of testing would most likely have exposed this defect?
    a. unit testing
    b. integration testing
    c. acceptance testing
    d. installation testing
    e. performance testing

67. The getBundle method in the Bundle component requires a pointer to an array to be passed as an argument, but a call to the getBundle method in the Reports component passes the value of an array instead. Which type of testing would most likely have exposed this defect?
    a. performance testing of the Bundle and Reports components
    b. installation testing of the Reports component
    c. unit testing of the Bundle component
    d. integration testing of the Bundle and Reports components
    e. acceptance testing of the Bundle and Reports components

68. The bundle selector is supposed to allow the analyst to choose manually whether or not a loan should be included in the bundle, but the manual modification of bundle selection has not been implemented. Which type of testing would most likely have exposed this defect?
    a. unit testing
    b. integration testing
    c. performance testing
    d. acceptance testing
    e. function testing

69. The Loan Arranger has to interface with another external banking system. The interface to the external system has not been specified correctly. Which type of testing would most likely have exposed this defect?
    a. integration testing
    b. installation testing
    c. performance testing
    d. acceptance testing
    e. function testing

70. The missing functionality described in question 68 is a serious defect in the system. It will take a significant amount of effort to correct the problem by doing redesign, recoding, and retesting. The team leader is not sure how this can be accomplished in the time remaining before the due date, or even if it is still possible to make that date. In trying to decide how to address this problem he thinks immediately of a system that was under development last year and also had significant missing functionality. The solution used on that system, of going straight to implementation of the new functionality without spending time on a redesign, might also work in this case, with some adaptations to take into account the fact that the current development team is much less experienced. Reasoning in this way is known as “anchoring and adjustment” and potential problems are that:
    a. The “anchoring” dominates and there is too little adjustment of the previous solution to the specific circumstances of the new problem.
    b. Arguing from analogy is always inherently dangerous.
    c. More suitable analogies might be overlooked because they are less recent.
    d. A and B
    e. A and C
    f. B and C
    g. A, B, and C

The bundle selection functionality is expected to be very complex, and there is concern about the reliability of this software. It is decided that faults will be seeded in the code to estimate the remaining faults. Two test teams will test the software.

Suppose 100 faults have been seeded in the code. During testing by the first team, 120 faults are detected. Sixty of the detected faults are seeded faults.

71. What is the Mills estimate for the percentage of remaining, non-seeded (indigenous) faults in the code?
    a. 10%
    b. 40%
    c. 50%
    d. 60%

    e. It is impossible to determine from the information given.

72. What is the Mills estimate of the total number of indigenous faults remaining?
    a. 7.5
    b. 10
    c. 40
    d. 50
    e. 60
    f. It is impossible to determine from the information given.

Suppose the same code is given to the second test team. This team finds 80 seeded faults and 70 non-seeded faults. 90 of the faults found by this team were also found by the other team.

73. Using the numbers for the second team, what is the Mills estimate for the total number of indigenous faults remaining?
    a. 0
    b. 17.5
    c. 20
    d. 87.5
    e. It is impossible to determine from the information given.

74. What is the effectiveness of the second test group?
    a. 30%
    b. 40%
    c. 60%
    d. 85%
    e. It is impossible to determine from the information given.

75. What is the effectiveness of the first test group?
    a. 25%
    b. 42%
    c. 64%
    d. 75%
    e. It is impossible to determine from the information given.

76. Based on the effectiveness of both test groups, what is the estimate for the total number of faults?
    a. 200
    b. 100
    c. 78
    d. 32
    e. It is impossible to determine from the information given.

77. Suppose 49 faults have been seeded into a component. Testing of the component has uncovered 45 of the seeded faults without uncovering any additional non-seeded faults. What is the level of confidence that the component is fault-free?
    a. 74%
    b. 80%
    c. 85%
    d. 92%
    e. None of the above.

After testing is complete, the system is delivered to the customer. Once it has been in operation for some time, a number of problem reports are returned.

78. Consider the following excerpts from problem reports filed for the Loan Arranger. In which type of report, discrepancy or fault, does each item belong? Answer fault report or discrepancy report.
    a. "The requirements document states that after the search for desirable loans returns a list of loans, the user should be able to remove or add loans from the list. The current software does not allow the user to modify the list after the search."
    b. "After submitting a search query for desirable loans, the results never came back. There should be a time out on the search and/or a message indicating the search is still in progress."
    c. "The search criteria are being ignored. There are two possible problems. The criteria may not be initialized correctly or the updates to the criteria may not be working properly. Check the constructor and the SetCriteria method of the Criteria class."

Other ideas for changes to the system are identified by the development team.

79. In order to deliver the system on time, the development team implemented a straightforward, brute force algorithm for creating bundles. This algorithm is sufficient for the consolidating organization's current needs but will not be able to meet the performance requirements if the volume of business increases. This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

80. If the brute force algorithm described in question 79 does cause the Loan Arranger system to not meet the performance requirements, this situation would be an example of:
    a. An exception
    b. An error
    c. A fault
    d. A failure
    e. None of the above

81. One of the banks has defects in its own software and sometimes sends data to the consolidating organization in which some of the records are not in the correct format. When this occurs the Loan Arranger alerts the loan analyst and none of the data from this bank is updated in the portfolio. This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

82. If the Loan Arranger receives bad data and reacts as described in question 81, this situation would be an example of:
    a. An exception
    b. An error
    c. A fault

    d. A failure
    e. None of the above

83. A mistake is noticed in the algorithm that computes the credit standing of loan recipients. That is, the value of a loan may be incorrectly computed because it is assumed to be a more or less risky proposition for investment than it actually is. This situation:
    a. Should lead to a corrective change.
    b. Should lead to an adaptive change.
    c. Should lead to a perfective change.
    d. Should lead to a preventive change.
    e. Should require no maintenance to be performed.

84. The erroneous module from question 83 (that computes credit standing) suffers from:
    a. An exception
    b. An error
    c. A fault
    d. A failure
    e. None of the above

85. Upon investigation, it is found that the problem described in question 83 results from a misconception in the original requirements. That is, the development team misunderstood the algorithm that the representative from the consolidating organization described. Then, this misunderstood algorithm was carried through the requirements, design, and implementation phases. This problem might have been discovered earlier if the team had used an appropriate:
    a. Linker
    b. Debugging tool
    c. Cross-reference generator
    d. Static code analyzer
    e. None of the above

Based on lessons learned during the Loan Arranger project, the team leader would like to invest further in process improvement.

86. Because he feels the need for continuing to improve the software development process, the team leader has decided to use CMM as a guide for process improvement. Which of the following represent potential problems that the team may encounter with CMM?
    a. It is not possible to customize the CMM to any special needs of the organization.
    b. The team might feel it necessary to invest in key process areas from a maturity level 2 or 3 levels higher than the organization's current ranking.
    c. The assumption behind the CMM is that every key process area is needed by the organization; in reality, this assumption might not be correct.
    d. None of the above represent real problems with the CMM.
    e. a and b
    f. a and c
    g. b and c
    h. a, b, and c

87. Which of the following are reasonable rationales for choosing CMM over another process maturity model?

a. Unlike SPICE, CMM clearly defines a set of desirable practices and processes.
b. Unlike in ISO9000, software measurement is a strong and explicit component of CMM.
c. Unlike both SPICE and ISO9000, the goals of CMM can be easily mapped to concrete questions and metrics.
d. a and b
e. a and c
f. b and c
g. a, b, and c

Final Exam Answers

1. d; Choice C is false because the requirements cover only what functionality is to be implemented, not how. [Section 4.1]
2. d; Data flow diagrams describe how data are input, processed, and output by the system but do not contain any mechanism for describing concurrency. However, SADT does allow multiple views of the system at different levels of detail, and Warnier diagrams do help organize the relationships among data. [Section 4.5]
3. TRUE; The requirements specification should contain anything relevant to how the system will interact with its environment. [Section 4.2]
4. FALSE; The requirements specification should contain anything relevant to how the system will interact with its environment. The requirements should describe what data are input to the system; why those data are of interest to the organization is outside the scope of the requirements. [Section 4.2]
5. TRUE; The requirements specification should contain anything relevant to how the system will interact with its environment. [Section 4.2]
6. TRUE; The requirements specification should contain anything relevant to how the system will interact with its environment. [Section 4.2]
7. e; Choice B is ambiguous because it mentions that some fields are changeable but does not mention which ones. [Section 4.3]
8. g; Nonfunctional requirements describe constraints on the system; typically, these constraints limit developers' choices in constructing the system. [Section 4.1]
9. b; This requirement is ambiguous, because if a borrower has both default and late loans it is not clear whether the borrower is in 'default' or 'late' status. [Section 4.3]
10. d; The phrase 'easily extensible' is unverifiable. How can extensibility be measured? [Section 4.3]
11. e; The formula given can be easily verified for correctness, and is not ambiguous. [Section 4.3]
12. b and d; This requirement is ambiguous because there is no definition given of “inappropriate” and “illegal”. As such, this condition cannot be tested, since it is unclear what set of inputs are intended to yield advice from the system. [Section 4.3]
13. TRUE; Lack of experience is a risk. [Section 3.4]
14. TRUE; Interfacing with externally developed systems is a risk. [Section 3.4]
15. FALSE; Prototyping is a risk control. [Section 3.4]
16. b; Predictions are noisy when they fluctuate more wildly than the actual measure.[Section 13.1]
17. b; (2*3) + (5*2) + (3*1) + (10*8) + (6*5) + (4*2) = 137 [Section 3.3]
18. c; $MMRE = \dfrac{\frac{5000}{10000}+\frac{5000}{20000}+\frac{30000}{80000}+\frac{40000}{110000}+\frac{5000}{45000}+\frac{5000}{50000}}{6} = 0.2825$ [Section 3.3]

19. d; Half of the estimates are within 25% of actual values. [Section 3.3]

20. No; using criteria *MMRE* < 0.25 and *PRED*(0.25) > 0.75. [Section 3.3]

21. a; 700 person-months is the average of the four estimates [Section 3.3]

22. FALSE; Although the method tends to produce convergence on the median estimate, a strong personality in the group (or other group dynamics issues) can push the group consensus toward another value. [Section 14.3]

23. e; 35 months [Section 3.3]

24. c; (n(n-1))/2)= 20(19)/2 = 190 lines of communication [Section 3.2]

The following table can be used to answer questions 25 to 28:

Activity	Earliest Start Time	Latest Start Time	Slack
A	1	1	0
B	8	13	5
C	6	6	0
D	10	15	5
E	9	9	0
F	16	21	5
G	19	19	0
H	23	23	0
I	24	28	4
J	33	33	0
K(finish)	37	37	0

An activity label in the table should be read, “the activity beginning at milestone $<label>$.” For example, the activity beginning at milestone B has an earliest start time of 8.

25. a; ACEGHJK is the critical path. In the table above, it represents the path with 0 slack time from start (A) to finish (K). [Section 3.1]

26. b; 5 is the slack time for the activity starting at milestone F. [Section 3.1]

27. d; 37 is the length of the critical path. [Section 3.1]

28. d; I is not a precursor to J [Section 3.1]

29. g; The conceptual design addresses issues such as what the system looks like to users, where the data comes from, and what happens to the data in the system. [Section 5.1]
30. e; The conceptual design should be able to be understood by the customer; it should therefore contain financial information but not implementation details. (There is not necessarily any connection between financial formulas and implementation.) The technical design should include more details about how the system is to be implemented. [Section 5.1]
31. b; If system functions are highly interrelated, it will be difficult to separate the system into components for which the internal organization and the relations to other components can be described. [Section 5.2]
32. f; Unlike those in pipe-and-filter systems, Object-Oriented components are not completely independent; certain changes to a component can require changes to all components that call it. [Section 5.3]
33. c; Design by contract views a software system as a set of communicating components, which may be a useful way to think of the components in the Object-Oriented design. Design by contract also places the most emphasis

on how components interact (specifying the preconditions, postconditions, and invariants that exist when one component calls another). Since the interaction of components is an area that will be important to address in this design, design by contract appears the best choice. [Section 5.6]

34. a; Coupling measures the amount of dependence among components. Since class Loan interacts with 4 other classes (at least double the number for any other class), class Loan has a high degree of coupling. [Section 5.5]
35. c; Cohesion measures how related the internal parts of a component are. Since this figure gives no details about the internal structure of components, no conclusions about cohesion can be drawn. [Section 5.5]
36. b; The high degree of coupling for class Loan leads to the possibility that a change to this class may require changes in many other parts of the system. A change to class Borrower, on the other hand, has the potential to affect only one other class. Additionally, components are often easier to understand if they are not intrinsically tied to others. Thus for many types of changes it is reasonable to assume that modifications to class Loan will be more difficult. [Section 5.5]
37. A. The high-level design should describe real-world entities in the problem, not the details of the solution. [Section 6.5]
38. B. Number of key classes can be measured during system design to get an idea of the size of the system, while number of support classes cannot be accurately measured until program design. The “lack of cohesion of methods” metric is useful for finding complex classes that can benefit from additional care in construction, not for estimating system size. [Section 6.7]
39. FALSE. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. [Section 6.7]
40. TRUE. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class. [Section 6.7]
41. TRUE. If a large number of methods can be invoked in response to a message, the testing and debugging of the class become more complicated since the class requires a greater level of understanding on the part of the tester. [Section 6.7]
42. A. According to the diagram, once in default status, the borrower can never return to good or late status. [Section 6.5]
43. FALSE. The black dot representing the start state leads to the state marked “Good Status,” meaning that it will be the default state for any new object of this type. [Section 6.5]
44. b; Common coupling exists when the design is organized such that data are accessible from a common data store, and potentially multiple components can access that data. [Section 5.5]
45. d; The design should describe the system in such a way that it can be validated whether the system will meet the needs of the organization. These needs should include not only day-to-day use but longer term needs such as archiving. [Section 5.8]
46. g; Program designers are present both to critique the design and to better understand it, so that they can then derive their more detailed program designs from it. If major problems are identified, the design is redone. [Section 5.7]
47. c; This study is a case study, since key factors that may affect the outcome are identified, documented, and controlled as much as possible. Since the study will be conducted on a single project that has real constrains and deadlines, we can assume that the level of control of key variables will not be high enough to make this a formal experiment. Since the project in which the new process is being evaluated will be compared

to a set of past projects that are meant to be typical, the case study makes use of a baseline for comparison purposes. [Section 12.1]

48. b; The aim in this type of study is to select a subset of past projects for comparison that are as similar as possible to the one using the new process. This selection process helps ensure that any differences are due to the new process and not other sources of variation. [Section 12.1]
49. b; The Loan Arranger is a P-type system, since the problem (tracking and bundling loans) can be described directly and completely, and has an exact solution. Unlike an E-type system, the system is not embedded in the environment, that is, the practical abstraction of the problem is unlikely to change due to an improved understanding resulting from the solution. As a P-type system, incremental change is possible in order to improve the solution. [Section 11.1]

50. error; These statements are describing a misconception on the part of the developer. [Sidebar 1.1]
51. failure; These statements describe a departure from the required behavior. [Sidebar 1.1]
52. fault; These statements describes a mistake that has been manifested in the code. [Sidebar 1.1]
53. failure; This is an example of the system performing incorrectly. [Sidebar 1.1]
54. a; This example is of producer reuse since reusable components are being created. The situation described also illustrates black-box reuse since the module is meant to be reused without modification. [Section 12.4]
55. g; The need to search through large repositories of components to find the best one for a particular reuse need is one of the biggest obstacles to effective reuse. Section 12.4 of the textbook describes some work in component classification that attempts to solve this problem. It also describes experiences at Raytheon and GTE, and how these organizations have designed their reuse programs to avoid the pitfalls described in choices B and C.

56. d; No documentation

57. b; The two loops can be combined to make this code more efficient.

58. c; The *else* clause matches with the first *if* clause. The formatting of this code makes it misleading.

59. b; Because the equation to calculate the profit is incorrect, the fault is a computation fault. [Section 8.1]

60. a; The variable *i* is initialized incorrectly. [Section 8.1]

61. c; list[10] is out of the defined array boundary [Section 8.1]

62. d; [Section 8.3]

63. a; top-down testing [Section 8.4]

64. b; bottom-up testing [Section 8.4]

65. c; sandwich testing [Section 8.4]

66. a; unit testing; This defect can be isolated to a single function in a single component. Unit testing should uncover this type of defect. [Section 8.2]

67. d; Since this defect involves the interface between the two components, integration testing of the Bundle and Reports components should detect the defect. Unit testing of the Bundle component alone would not uncover the defect since the defect exists in the Reports component. [Section 8.2]

68. e; Function testing is used to determine if the functions described in the requirements specification are actually implemented in the system. [Section 8.2]

69. a; Since the defect deals with interfaces, integration testing should detect the defect. [Section 8.2]

70. E; Arguing from analogy is a particularly useful way of learning from past experiences, but care must be taken not to "anchor" on the wrong past experience or to insufficiently "adjust" to the new circumstances. [Section 14.3]

71. b; The percentage of indigenous faults remaining is equal to the percentage of seeded faults remaining. (1 - 60/100) = .4 [Section 8.8]

72. c;

$$\frac{seeded\_faults\_found}{seeded\_faults} = \frac{indigenous\_faults\_found}{indigenous\_faults}$$

$$indigenous\_faults = \frac{seeded\_faults \times indigenous\_faults\_found}{seeded\_faults\_found}$$

$$indigenous\_faults = \frac{100 \times 60}{60} = 100$$

indigenous_faults_remaining = 100 – 60 = 40
[Section 8.8]

73. b;
indigenous faults = 100*70/80 = 87.5
indigenous faults remaining = 87.5 - 70 = 17.5
Section 8.8

74. c; effectiveness = overlapping faults/faults found by the second group
effectiveness = 90/150 = 60%
[Section 8.8]

75. d; effectiveness = 90/120 = 75% [Section 8.8]

76. a; total faults = 90/(.6 * .75) = 200 [Section 8.8]

77. e;

$$C = \frac{\binom{49}{44}}{\binom{50}{45}} = 0.9$$

[Section 8.8]

78.

a) discrepancy report; This description describes a difference between the requirements and the implementation. [Section 9.8]
b) discrepancy report; This description describes a problem from the user's point of view. [Section 9.8]
c) fault report; The description of the problem includes information from the developer's point of view. [Section 9.8]

79. d; This change involves modifying part of the system to prevent future faults. [Section 11.2]
80. d; A failure is an instance during system operation in which system behavior deviates from expectations. [Section 5.5]
81. e; This type of response may or may not be the optimal way of handling such a situation (since discarding the entire report potentially discards valid records). However, since the response is consistent and reliable, it does not represent a defect in the system unless it somehow fails to meet the needs of the customer. [Section 11.2]
82. a; The description provided illustrates the Loan Arranger handling an exception, that is, responding to a situation that is counter to the intended operation of the system. [Section 5.5]
83. a; This change is necessary to directly correct a fault. [Section 11.2]
84. c; A fault is a defect in a software product, resulting from some human error. [Section 5.5]
85. e; Choices A through D are all tools that can help catch defects introduced after the requirements stage. To catch the faulty algorithm, a way of ensuring requirements correctness would have been necessary (for example, requirements reviews). [Section 11.5]
86. f; Both choices A and C are true statements about the CMM. The same key process areas are recommended for every organization. [Section 12.5]
87. b; Choice B is the only true statement. SPICE also defines a specific set of desirable practices, and both SPICE and CMM have goals that can be mapped to questions and metrics. [Section 12.5]