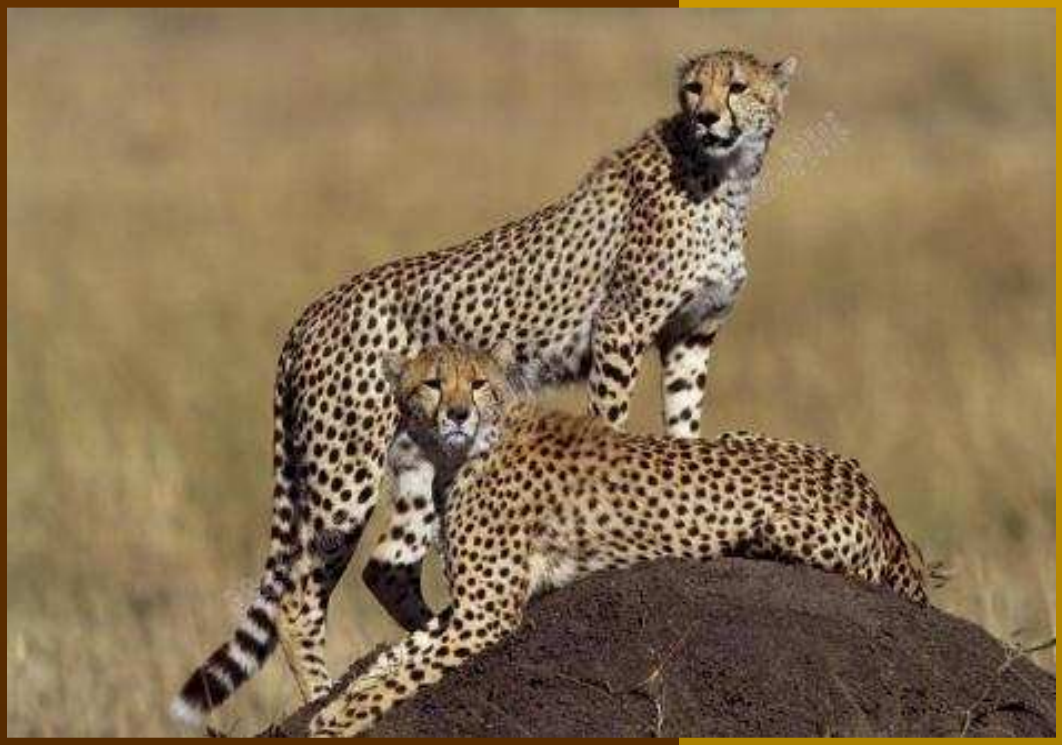
David Kung



Object-Oriented Software Engineering

An Agile Unified Methodology

Solutions Manual

Message to Instructors

July 10, 2013

The solutions provided in this manual may not be complete, or 100% correct, due to my limitation and the nature of some software engineering problems. Although I have tried to check and correct grammar errors and typos, I am sure the manual may still have many. I will continue to improve it and apologize for any inconvenience that may cause to you.

Dave

Contents

Part I

INTRODUCTION AND SYSTEM ENGINEERING

Chapter 1

Introduction

1.1 Search the literature and find four other definitions of software engineering in addition to the one given in this chapter. Discuss the similarities and differences between these definitions.

Solution. Below are five definitions of software engineering including the one in the textbook, listed chronologically. The similarities and differences are shown in Figure 1.1. A better solution should also provide a convincing explanation of the differences, and significant implications of the differences. For example, software engineering education, and significant improvement of software PQCT are important considerations of software engineering.

1. **IEEE.** The IEEE Computer Society defines software engineering as: "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)." ("IEEE Standard Glossary of Software Engineering Terminology," IEEE std 610.12-1990, 1990.)

2. **Ghezzi.** "Software engineering is the field of computer science that deals with the building of software systems that are so large or so complex that they are built by a

	IEEE	Ghezzi	Brugge	Sommerville	Kung
(1) Application of engineering to software	√	√		√	√
(2) Study of approaches as in (1)	√				√
(3) Modeling, problem-solving, knowledge acquisition, and rationale driven activity	√	√	√	√	√
(4) Education of engineering processes and methodologies					√
(5) Significantly improve software PQCT (that is, engineering and business aspects)					√

Figure 1.1: Similarities and differences between SE definitions

team or teams of engineers." It is "the application of engineering to software." (Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli, "Fundamentals of Software Engineering," 2nd Edition, Prentice Hall, 2003.)

3. **Brugge and Dutoit.** Software engineering is a *modeling, problem-solving, knowledge acquisition*, and *rationale-driven* activity. (Bernd Brugge and Allen H. Dutoit, "Object-Oriented Software Engineering Using UML, Patterns, and Java," 3rd Edition, Prentice Hall, 2010.)

4. **Sommerville.** "Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use." (Ian Sommerville, "Software Engineering," 9th Edition, Addison-Wesley, 2011.)

5. **Kung.** "Software engineering as a discipline is focused on the research, education, and application of engineering processes and methods to significantly increase software productivity and software quality while reducing software costs and time to market." (David Kung, "Object-Oriented Software Engineering: An Agile Unified Methodology," McGraw-Hill Higher Education, 2013.)

1.2 Describe in a brief article the functions of software development process, software quality

assurance, software project management, and software configuration management. Discuss how these work together during the software development life cycle. Discuss how they improve software PQCT.

Solution. This sample solution includes the main points. A student's solution may expand on issues discussed here.

"A software development process transforms an initial system concept into an operational system running in the target environment. Its functions include identification of business needs, conducting feasibility study, formulating capabilities that the system must deliver as well as design, implementation, testing and deployment of the system to the target environment. The functions of software quality assurance (SQA) include definition of quality assurance standards and procedures, and verification, validation and testing activities to ensure that the development process is carried out properly, and the software artifacts produced by the development activities meet the software requirements and desired quality standards. Software project management oversees the control and administration of the development and SQA activities. Its functions include effort estimation, project planning and scheduling, risk management, and project administration. These activities ensure that the software system is delivered on time and within budget. During the development process, numerous software artifacts are produced including software requirements specification (SRS), software design, code, test cases, user's manual, etc. These are the software, or part of it, under different stages of the development process. These documents depend on each other. This implies that changes to one document may affect other documents, which may need changes as well. Software configuration management (SCM) is a mechanism to coordinate changes to ensure that changes are made consistently and cost-effectively.

All of software development process, SQA, project management and SCM contribute to

PQCT. In particular, good software development practices would apply well-established software development methodologies, software design principles, software design patterns, coding standards, test-driven development. These could lead to improvement of software productivity and software quality while at the same time reduce software costs and time to market. SQA ensures that the software meets the requirements and quality standards. It contributes to improvement of software quality. This in turn reduces rework and field-detected bugs; and hence, it also improves software productivity, reduces costs associated with rework and fixing field-detected bugs. Software project management ensures proper planning and administration of the software project. In particular, it should request the needed resources to develop the software system, properly schedule the development activities and SQA activities, manage budget and risks. These indirectly contribute to improvement of software productivity and software quality. Proper planning and administration of development and SQA activities directly contribute to reducing software development costs and time to market. This is because these activities could be performed smoothly, e.g., the needed components and resources are in place. SCM supports project management, SQA and software development process. It ensures that components of the software system are constructed and modified consistently and cost-effectively. Consistent modification implies productivity and quality, and cost-effectiveness implies reduction in cost and time to market.

A student's answer to this question may also include a discussion of the balance between PQCT. See Exercise 1.5."

1.3 Should optimization be a focus of software engineering? Briefly explain, and justify your answer with a practical example.

Solution. The answer to this question may depend on the interpretation of "optimization."

If it is about "optimization of software PQCT," then it is the focus of software engineering. If it is about performance optimization, then it should not be a focus, although SE also considers performance issues such as testing for performance. The database access example discussed in Section 1.5 is a practical example.

Optimization could be a focus for a given project. For example, the construction of a compiler for multicore computers. In this case, it depends on whether the project is classified as a software engineering, or a computer science project. It might be an SE project. For example, it is constructed for a certain application. (See solution to Exercise 1.6 for more on optimization and SE.)

1.4 Identify three computer science courses of your choice. Show the usefulness of these courses in the software life-cycle activities.

Solution. An Algorithms and Data Structures course is useful in the implementation phase for the design and implementation of algorithms and data structures to implement business processes. In particular, the course lets the student know the available algorithms and related data structures. The computational complexity lets the student select appropriate algorithms and data structures according to the nature of the computation.

A Database Systems course is useful in the analysis, design, and implementation phases. In the analysis phase, it helps the student understand database related requirements such as the need to support multiple databases for some applications. In the design phase, the course enables the student to design the database to fulfill the requirements and constraints. In the implementation phase, the course provides the student abilities to store and retrieve information with a database.

A Discrete Mathematical Structures course is useful in many phases of the life cycle. In

particular, mathematical logic lets the student design and implement logically consistent and logically complete algorithms, and check for such properties during design review and code review. Graph theory helps the student understand design diagrams such as UML diagrams because UML diagrams are directed graphs. Thus, concepts and algorithms of graph theory can be applied. Examples include fan-in and fan-out of a class, transitive closure for computing the change impact of a class, traversal algorithms for calculating reachability in a state diagram.

Courses on programming languages are useful for implementation, testing, and SQA activities. Network courses are helpful in SE project that must communicate with a remote computer, such as accessing a remote database. Artificial intelligence courses are useful for SE projects that involve heuristic, and/or learning algorithms.

1.5 There are interdependencies between software productivity, quality, cost, and time to market. For example, more time and effort spent in coding could increase productivity. This may result in less time and effort in software quality assurance because the total time and effort are constants. Poor quality could cost productivity due to rework. Identify three pairs of such interdependencies of your choice. Discuss their short-term and long-term impacts on the software development organization. How should software engineering solve the "dilemmas" induced by the interdependencies?

Solution. Barry Boehm in his papers on software engineering economics pointed out that the cost to fix a requirements error increases exponentially with time. That is, removing errors as early as possible is a cost-saving effort. This also coincides with the philosophy advocated by agile methods — that is, test early and often. Thus, if SQA is carried out as a life-cycle activity and follows a good SQA process, then it should detect requirements, design and

implementation errors early. This reduces bug fixing costs exponentially. Moreover, since SQA is a cooperate-wide practice, developers are conscious of developing quality software. This should significantly reduce the error rate; and hence, it reduces the cost that would otherwise have to be spent to fix the bugs.

In the short term, implementing and executing an SQA framework may reduce productivity, increase costs and time to market. However, in the long term, quality software brings many benefits to the organization. These include significant reduction in error rate and error correction costs, customer satisfaction, and increase in software capability maturity level. These should positively impact productivity, cost and time to market.

One should be aware that quality is not the more the better. To a certain point, there is the so-called "diminishing returns." Thus, how much SQA is appropriate remains a research problem in general and for a software organization in particular.

1.6 What are the differences and relationships between OO software engineering and conventional software engineering? Discuss whether OO software engineering will replace conventional software engineering.

Solution. The main difference between conventional software engineering and OO software engineering is paradigm shift — that is, how they view the world and systems. Because of this, they differ in the basic concepts, basic building blocks, and starting point for the conceptualization, design and implementation of software systems. These in turn affect SQA, project management (for example, effort estimation and planning), and SCM.

Will OO replace the conventional paradigm? The answer should be no[1]. The reasons are:

[1]A student's solution may indicate "yes," and provide convincing arguments. Such a solution should also be considered a good solution.

1. There are numerous systems that were developed using one or more of the conventional paradigms. It is very costly and risky to replace these systems. Therefore, the other paradigms will continue to exist because bug fixing, performance improvements, and functional enhancements to these systems are required.

2. There are hundreds of thousand organizations and millions of software developers using only the conventional paradigms. It is practically impossible and unjustifiable to require them to convert to the OO paradigm.

3. A conventional paradigm may be more suitable for some projects. For example, scientific computing typically involves series of transformations of input into output. Therefore, the function-oriented paradigm is more suitable for such applications. Moreover, scientific computing emphasizes computing speed, the ability to solve complex computation problems, and the accuracy of the result. OO programming languages may not satisfy such requirements. These and the facts that scientific computing is there to stay and expand into computational sciences imply that the function-oriented paradigm will continue to exist.

In addition to the above, one should know that different parts of a system may be developed using different paradigms. For example, a subsystem that performs scientific computing may be developed using the function-oriented paradigm. A database subsystem may be developed using the data-oriented paradigm. In practice, there are systems that are modeled and designed using the OO paradigm but implemented in a non-OO language. Similarly, there are projects that are modeled and designed using a conventional paradigm but implemented in SmallTalk or C++.

Chapter 2

Software Process and Methodology

2.1 What are the similarities and differences between the conventional waterfall model and the Unified Process model? Identify and explain three advantages and three disadvantages of each of these two models.

Solution. The waterfall model and the Unified Process (UP) model are similar in the sense that they are process models, they define phases, the activities and products of each of the phases. The waterfall process is a sequential process although backtracking is possible. The UP, on the other hand, is an iterative, incremental process.

Waterfall process advantages are: (1) it facilitates project management, scheduling and status tracking, (2) its can be used for function-oriented team organzation, and (3) it is more appropriate for some types of software project. Its disadvantages are: (1) it is difficult to respond to requirements change, (2) the long development duration is unacceptable, and (3) users cannot experiment with the system until late in the development life cycle.

UP advantages are: (1) its iterative process can better accommodate requirements change because changes can be made to remaining iterations, (2) it is use-case driven, allowing the development team to focus on customer value — that is, development and deployment of

high-priority use cases as early as possible, (3) it is incremental, this reduces the risk of requirements misconception. It disadvantages are: (1) an iterative process is more difficult to manage and schedule, (2) the early versions of the UP emphasize too much on documentation and much of it is not used, (3) the UP is a process, not a methodology, therefore, it is useful only for experienced software developers.

2.2 Write an essay about how a good process and a good methodology help tackling the project and product challenges. Limit the length of the essay to five pages, or according to the instructions of the instructor.

Solution. There could be many different answers to this exercise. It is difficult to come up with a standard solution and use it to grade the submissions. However, the answer should show how a good process and methodology address each of the challenges. Figure 2.1 of this manual highlights the main points and provides pointers to related chapters.

Grading of this exercise could be done by reading the solutions submitted by the students, according to the writing, the grader classifies the solutions into 3-5 categories such as very good, good, fair, below, and poor. Each of the categories is then reviewed and a score is assigned to each of the solution.

2.3 Write a brief essay on the differences between a software process and a software methodology.

Solution. Figure 2.11 of the textbook shows the differences between a process and a methodology. Therefore, the student needs only to explain the differences in the essay. Section 2.6.1 of the textbook presents the differences. A student's solution may reuse the materials in the section, and/or augment with practical examples, or experience.

2.4 Write an essay that discusses the following two issues:

	Description	Process or Methodology Solutions
Project Challenge 1	How do we plan, schedule and manage a project without sufficient knowledge about what will happen in the next several years?	• Effort estimation, and project planning and scheduling (Chapter 23) • Agile planning (Chapter 23) • Agile manifesto, principles, practices, and values (Chapter 2, and throughout the book) • Agile development, i.e., design for change, frequent delivery of small increments in short intervals (various chapters) • Risk management (Chapter 23)
Project Challenge 2	How do we divide the work among different departments and teams, and smoothly integrate the resulting components?	• System engineering (Chapter 2) • Software architectural design, behavioral design, and derivation of design class diagram (Chapters 6-16) • Peer review, inspection and walkthrough (Chapter 18) • Integration testing (Chapter 19)
Project Challenge 3	How do we ensure proper communication and coordination among the departments and teams?	• Modeling, analysis, and design using a unified modeling language such as UML (various chapters) • Applying design patterns during the design process (Chapters 11 and 16) • Software configuration management (Chapter 22)
Product Challenge 1	How do we develop the system to ensure that these requirements and constraints are met?	• Deriving use cases from requirements (Chapter 5) • Use case driven (various chapters) • Peer review, inspection and walkthrough (Chapter 18) • Acceptance testing and system testing (Chapter 19)
Product Challenge 2	How do we cope with changes?	• Design for change (various design chapters) • Applying design patterns to provide the needed flexibility (Chapters 11 and 16)
Product Challenge 3	How do we design the system so that changes can be made relatively easily and without much impact to the rest of the system?	Same as product challenge 2
Product Challenge 4	How do we design the system to hide the hardware, platform and implementation so that changes to these will not affect the rest of the system?	Same as product challenge 2

Figure 2.1: Dealing with project and product challenges

a. The pros and cons of plan-driven development and agile development processes, respectively.

b. Whether and why agile development will, or will not, replace plan-driven approaches.

Solution. The solution to 2.4a is similar to the solution about the differences between the waterfall and UP process models. The answer to 2.4b can be "yes" or "no," and the answer is not that important. The importance is the understanding of the differences between the two approaches, and the student's reasoning to justify the conclusion. This exercise should be graded using the method described in the solution for Exercise 2.2.

2.5 Write a short article that answers the following questions:

a. What are the similarities and differences between the spiral process, the Unified Process, and an agile process.

b. What are the pros and cons of each of these processes.

c. Which types of projects should apply which of these processes?

Solution. The similarities are that they are iterative processes, and meant to be an improvement over the existing processes. However, the iterations in the spiral process is situation dependent — that is, what to perform next depends on the outcome of the current iteration. Moreover, risk management is a unique feature of the spiral process. Unlike the spiral process, the UP repeats the same four phases in each iteration. It does not require the spiral process like decision making. It also does not indicate risk management. Agile processes are different from the spiral and UP in the agile manifesto, agile practices and values, and agile principles. In addition, agile development tend to adopt short iterations and frequent delivery of small increments. There are other differences but a solution should focus on these.

Among the three choices, projects that are research-oriented or exploratory may use the

spiral process. Projects that require adequate documentation should use the UP. Projects that need to respond quickly to changing business environments, and hence software requirements, should use an agile process. There are subtle differences between "research-oriented" and "changing requirements." Both need to tackle changing requirements. Research-oriented requirements need to be discovered with research tasks and experiments, which require considerable time and effort, and the costs are high.

2.6 Explain in an essay why the waterfall process is a process for solving tame problems.

Solution. The waterfall process requires that the requirements of the system must be identified, clearly and completely defined before the design and implementation of the system. This is at least true in theory, although many real-world projects do not happen like this. The first two properties of wicked problems are: (1) a wicked problem does not have a definite formulation, and (2) the specification of the problem and the solution cannot be separated. Clearly, the waterfall process cannot solve wicked problems because the problem-solving process does not address these two wicked-problem properties. A student's solution may address other properties as well. (See also solution to Exercise 2.7, especially Figure 2.2 of this manual. From the discussion and the Figure 2.2, one may infer more on the inadequacy of waterfall in solving software development as a wicked problem.)

2.7 Explain in an essay how agile development tackles application software development as a wicked problem.

Solution. Software development as a wicked problem implies that the requirements for a software system cannot be completely and definitely formulated, and the specification and the solution cannot be separated — the specification is the solution, and vice versa. Agile development recognizes these and advocates responding to requirements change. The 20/80

Properties of a Wicked Problem	Agile Development Solution
1) A wicked problem does not have a definite formulation.	• Value working software over comprehensive documentation. • Value responding to change over following a plan. • Capture requirements at a high level, lightweight, and visual. • User involvement is imperative. • Good enough is enough.
2) For a wicked problem, the specification is the solution and vice versa.	
3) There is no stopping rule for a wicked problem --- you can always do it better.	• Requirements evolve but the timescale is fixed. • Don't work over 40 hours a week.
4) Solutions to wicked problems can only be evaluated in terms of good or bad, and the judgment is subjective.	• User involvement is imperative. • The team is empowered to make decisions. • Users perform testing.
5) Each step of the problem-solving process has an infinite number of choices ---everything goes as a matter of principle.	• Value individual and interaction over processes and tools. • The team is empowered to make decisions.
6) Cause-effect reason is premise-based, leading to varying actions, but hard to tell which one is the best.	• User involvement, value individual and interaction, team decision making. • A collaborative and cooperative approach between all stakeholders is essential. • Value customer collaboration over contract negotiation.
7) The solution cannot be tested immediately and is subject to life-long testing.	
8) Every wicked problem is unique.	
9) The solution process is a political process.	
10) The problem-solver has no right to be wrong because the consequence is disastrous.	

Figure 2.2: Wicked problems and agile development as a solution

rule indicates that it is good enough to identify 80% of the requirements that are of high customer value. In additiion, it advocates capturing requirements at a high level, lightweight, and visual. That is, low-level requirements are to be captured during the implementation phase. This is because the specification and the solution cannot be separated. Agile development also emphasizes on user involvement because the "correctness" of a software system cannot be determined objectively and scientifically. Figure 2.2 of this manual shows how agile manifesto and principles solve wicked problems.