Software Engineering 10

Solutions Manual

IAN SOMMERVILLE

These solutions are made available for instructional purposes only. Neither the author nor the publisher warrants the correctness of these solutions nor accepts any liability for their use. Solutions may only be distributed to students and it is a condition of distribution that they are only distributed by accredited instructors using 'Software Engineering, 10^{h}edition' as a textbook. The solutions may be made available to students on a password-protected intranet but must not be made available on a publicly-accessible WWW server.

©Ian Sommerville 2014

Preface

This solutions manual is intended to help teachers of software engineering courses in marking homework questions for students. Each chapter in the book has 10 exercises of different types, which you may set for students either as is or in a modified form. I have supplied answers to 50% of the exercises in this manual.

The exercises for which answers have not been supplied are, generally, of one of three types:

1. Simple exercises whose answers can be found in the text of the chapter. There are typically one or two of these questions in each chapter and they are intended to stimulate students to read the chapter.

2. Design problems for which there is a range of solutions and you have to use your judgment to decide if the solution is appropriate. Supplying a solution here would imply that there is only one right answer to the question.

3. Ethics-related questions as the aim of these questions is to encourage students to think about the ethics issues involved. The notion of a right and wrong answer does not apply in this case as the student's response to the question depends both on their cultural background and on their particular views on a topic. I suggest that these questions should be used to stimulate class discussions rather than as part of class tests.

It is important when marking the student's answers to exercises to see the supplied solutions as a guide only rather than a definitive statement of the only possible answer to the question. It is generally good educational practice to give students credit for what they know and if they produce credible answers that reveal they have thought about the exercise and have some knowledge of the topic, then this should be rewarded.

This solutions manual may be used in conjunction with the associated quiz book, which lists short questions and answers for each chapter in the book. These can be used for short class tests to assess if students have read the material or as self-assessment tests which the students complete in their own time.

If you think that I have made a mistake in some of these answers (quite possible), please let me know. In some cases, there are obviously several possible answers and you may disagree with my solutions. I'd be delighted to consider including your alternative solutions but I do not have time to engage in detailed email discussions about the exercises in the book.

Ian Sommerville
October 2014

©Ian Sommerville 2014

1 Introduction

1.2 What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?

The essential difference is that in generic software product development, the specification is owned by the product developer. For custom product development, the specification is owned and controlled by the customer. The implications of this are significant – the developer can quickly decide to change the specification in response to some external change (e.g. a competing product) but, when the customer owns the specification, changes have to be negotiated between the customer and the developer and may have contractual implications.

For users of generic products, this means they have no control over the software specification so cannot control the evolution of the product. The developer may decide to include/exclude features and change the user interface. This could have implications for the user's business processes and add extra training costs when new versions of the system are installed. It also may limit the customer's flexibility to change their own business processes.

1.3 What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant.

Four important attributes are maintainability, dependability, performance and usability. Other attributes that may be significant could be reusability (can it be reused in other applications), distributability (can it be distributed over a network of processors), portability (can it operate on multiple platforms e.g laptop and mobile platforms) and inter-operability (can it work with a wide range of other software systems).

Decompositions of the 4 key attributes e.g. dependability decomposes to security, safety, availability, etc. is also a valid answer to this question.

©Ian Sommerville 2014

1.4 Apart from the challenges of heterogeneity, business and social change and trust and security, identify other problems and challenges that software engineering is likely to face in the 21st century (hint: think about the environment).

Problems and challenges for software engineering

There are many possible challenges that could be identified. These include:

1. Developing systems that are energy-efficient. This makes them more usable on low power mobile devices and helps reduce the overall carbon footprint of IT equipment.
2. Developing validation techniques for simulation systems (which will be essential in predicting the extent and planning for climate change).
3. Developing systems for multicultural use
4. Developing systems that can be adapted quickly to new business needs
5. Designing systems for outsourced development
6. Developing systems that are resistant to attack
7. Developing systems that can be adapted and configured by end-users
8. Finding ways of testing, validating and maintaining end-user developed systems

1.5 Based on your own knowledge of some of the application types discussed in section 1.1.2, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.

Different application types require the use of different development techniques for a number of reasons:

1. Costs and frequency of change. Some systems (such as embedded systems in consumer devices) are extremely expensive to change; others, must change frequently in response to changing requirements (e.g. business systems). Systems which are very expensive to change need extensive up-front analysis to ensure that the requirements are consistent and extensive validation to ensure that the system meets its specification. This is not cost-effective for systems that change very rapidly.
2. The most important 'non-functional' requirements. Different systems have different priorities for non-functional requirements. For example, a real-time

©Ian Sommerville 2014

control system in an aircraft has safety as its principal priority; an interactive game has responsiveness and usability as its priority. The techniques used to achieve safety are not required for interactive gaming; the extensive UI design required for games is not needed in safety-critical control systems.

3. The software lifetime and delivery schedule. Some software systems have a relatively short lifetime (many web-based systems), others have a lifetime of tens of years (large command and control systems). Some systems have to be delivered quickly if they are to be useful. The techniques used to develop short-lifetime, rapid delivery systems (e.g. use of scripting languages, prototyping, etc.) are inappropriate for long-lifetime systems which require techniques that allow for long-term support such as design modelling.

1.8 Discuss whether professional engineers should be certified in the same way as doctors or lawyers.

These are possible discussion points - any discussion on this will tend to be wide ranging and touch on other issues such as the nature of professionalism, etc.

Advantages of certification

- Certification is a signal to employers of some minimum level of competence.
- Certification improves the public image of the profession.
- Certification generally means establishing and checking educational standards and is therefore a mechanism for ensuring course quality.
- Certification implies responsibility in the event of disputes. Certifying body is likely to be accepted at a national and international level as 'speaking for the profession'.
- Certification may increase the status of software engineers and attract particularly able people into the profession.

Disadvantages of certification

- Certification tends to lead to protectionism where certified members tend not to protect others from criticism.
- Certification does not guarantee competence merely that a minimum standard was reached at the time of certification.
- Certification is expensive and will increase costs to individuals and organisations.
- Certification tends to stultify change. This is a particular problem in an area where technology developments are very rapid.

©Ian Sommerville 2014

2 Software Processes

2.1 Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car
- A virtual reality system to support software maintenance
- A university accounting system that replaces an existing system
- An interactive travel planning system that helps users plan journeys with the lowest environmental impact

1. *Anti-lock braking system* This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analysed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.
2. *Virtual reality system* This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.
3. *University accounting system* This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.
4. *Interactive travel planning system* System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

2.3 Consider the integration and configuration process model shown in Figure 2.3. Explain why it is essential to repeat the requirements engineering activity in the process.

©Ian Sommerville 2014

You need to repeat the requirements engineering activity because it is essential to adapt the system requirements according to the capabilities of the system/components to be reused. These activities are:

1. An initial activity where you understand the function of the system and set out broad requirements for what the system should do. These should be expressed in sufficient detail that you can use them as a basis for deciding of a system/component satisfies some of the requirements and so can be reused.
2. Once systems/components have been selected, you need a more detailed requirements engineering activity to check that the features of the reused software meet the business needs and to identify changes and additions that are required.

2.4 Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

There is a fundamental difference between the user and the system requirements that mean they should be considered separately.

1. The user requirements are intended to describe the system's functions and features from a user perspective and it is essential that users understand these requirements. They should be expressed in natural language and may not be expressed in great detail, to allow some implementation flexibility. The people involved in the process must be able to understand the user's environment and application domain.
2. The system requirements are much more detailed than the user requirements and are intended to be a precise specification of the system that may be part of a system contract. They may also be used in situations where development is outsourced and the development team need a complete specification of what should be developed. The system requirements are developed after user requirements have been established.

2.6 Explain why change is inevitable in complex systems and give examples (apart from prototyping and incremental delivery) of software process activities that help predict changes and make the software being developed more resilient to change.

Systems must change because as they are installed in an environment the environment adapts to them and this adaptation naturally generates new/different

©Ian Sommerville 2014

system requirements. Furthermore, the system's environment is dynamic and constantly generates new requirements as a consequence of changes to the business, business goals and business policies. Unless the system is adapted to reflect these requirements, its facilities will become out-of-step with the facilities needed to support the business and, hence, it will become less useful.

Examples of process activities that support change are:

1. Recording of requirements rationale so that the reason why a requirement is included is known. This helps with future change.
2. Requirements traceability that shows dependencies between requirements and between the requirements and the design/code of the system.
3. Design modeling where the design model documents the structure of the software.
4. Code refactoring that improves code quality and so makes it more amenable to change.

2.9 Suggest two advantages and two disadvantages of the approach to process maturity that is embodied in the SEI's Capability Maturity Framework.

Advantages of process improvement frameworks

1. The approach provides a means of measuring the state of a process and a structured approach to introducing process improvements.
2. It is useful as a way of building on the experience of others in process improvement.

Disadvantages of process improvement frameworks

1. Like any measurement system, there is a tendency to introduce improvements to improve the measured rating rather than concentrate on improvements that meet real business goals.
2. The maturity model approach is expensive and bureaucratic to operate. It is not really suitable for organisations that use agile development.

©Ian Sommerville 2014

3 Agile Software Development

3.2 Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

1. *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team know what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.
2. *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.
3. *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.
4. *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

©Ian Sommerville 2014

3.3 Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Advantages of stories:

1. They represent real situations that commonly arise so the system will support the most common user operations.
2. It is easy for users to understand and critique the stories.
3. They represent increments of functionality – implementing a story delivers some value to the user.

Disadvantages of stories

1. They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
2. They focus on functional requirements rather than non-functional requirements.
3. Representing cross-cutting system requirements such as performance and reliability is impossible when stories are used.
4. The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

3.6 Compare and contrast the Scrum approach to project management with conventional plan-based approaches as discussed in Chapter 23. Your comparison should be based on the effectiveness of each approach for planning the allocation of people to projects, estimating the cost of projects, maintaining team cohesion and managing changes in project team membership.

Planning allocation of people to projects

Scrum

Scrum handles people allocation informally. Team members 'bid' for features from the product backlog to implement if they think that their expertise is appropriate. Alternatively, the tasks can be allocated by the Scrum master.

There is no formal mechanism in Scrum for planning for project members with very specific expertise to be temporarily allocated to a team. This need must be identified by the Scrum master and he or she has to discuss how the expertise can be made available.

Plan-based development

©Ian Sommerville 2014

A project plan is used to identify the parts of the system to be delivered and these are specified in the requirements document. The expertise required for each part can then be identified and the allocation of people to projects planned on that basis.

Estimating project costs

Scrum

Project costs are estimated based on the required delivery date for the software and people working in the Scrum team. The functionality of the system is adjusted so that some working system will always be delivered for the original cost estimation. Of course, this may not be adequate for the customer and they have to become involved in rescheduling the delivery of the system.

Plan-based development

Project costs are based on an analysis of the functionality specified in the requirements document as well as the non-functional requirements of the system. They may be adjusted to reflect team size and delivery schedule. It is normal for costs to be underestimated and the final project to cost much more than originally estimated. An average cost for team members is assumed.

Maintaining team cohesion

Scrum

Team member meet daily either face to face or electronically. Extensive informal discussions and communications are encouraged. Team members negotiate work to be done from the project backlog. This all leads to a shared feeling of product ownership and a very cohesive team.

Plan-based development

Team cohesion is the responsibility of the project manager and he or she has to take explicit actions to encourage this. The general approach relies on formal meetings that are relatively infrequent and this does not lead to the development of a cohesive team.

Managing changes in project team membership

Scrum

This is a topic that is rarely discussed in Scrum but is a fundamental problem because so much information is informal and reliant on people remembering what has been agreed. When someone leaves, it can be very difficult to bring a replacement team member up to speed, especially if very little project documentation is available.

Plan-based development

©Ian Sommerville 2014

The project management plan is based around expertise rather than individuals and project documents should be available. Therefore, if a team member leaves, then a new team member with comparable expertise can read what has been done and, after understanding this, should be able to serve as a replacement.

3.8 Why is it necessary to introduce some methods and documentation from plan-based approaches when scaling agile methods to larger projects that are developed by distributed development teams.

---

1. *Project planning* is often essential when developing software with larger teams to (a) ensure that the right people are available when they are needed to be involved in the development process and (b) ensure that the delivery schedules of different parts of the system developed by different teams are aligned. This means that if Part A depends on Part B, the schedule should ensure that Part B is developed before Part A.
2. *Requirements analysis and documentation* is important to decide how to distribute the work across teams and to ensure that each team has some understanding of what other teams are doing.
3. *Design documentation* especially interface specifications are important so that teams can develop independently without having access to software that is under development.
4. *Risk management* may be required to ensure that all of the teams understand the risks faced and can organize their work to minimize these risks. Risk management may also be useful to cope with different delivery schedules used by different teams.

3.10 It has been suggested that one of the problems of having a user closely involved with a software development team is that they 'go native'. That is, they adopt the outlook of the development team and lose sight of the needs of their user colleagues. Suggest three ways how you might avoid this problem and discuss the advantages and disadvantages of each approach.

---

1. *Involve multiple users in the development team.* Advantages are you get multiple perspectives on the problem, better coverage of user tasks and hence requirements and less likelihood of having an atypical user. Disadvantages are cost, difficulties of getting user engagement and possible user conflicts.

©Ian Sommerville 2014

2. *Change the user who is involved with the team*. Advantages are, again, multiple perspectives. Disadvantages are each user takes time to be productive and possible conflicting requirements from different users.
3. *Validate user suggestions with other user representatives*. Advantages are independent check on suggestions; disadvantage is that this slows down the development process as it takes time to do the checks.

©Ian Sommerville 2014

4 Requirements Engineering

4.2 Discover ambiguities or omissions in the following statement of requirements for part of a ticket-issuing system:

An automated ticket machine sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination and the type of ticket required. Once a destination has been selected, the ticket price is displayed and customers are asked to input their credit card. Its validity is checked and the user is then asked to input their personal identifier (PIN). When the credit transaction has been validated, the ticket is issued.

Ambiguities and omissions include:

1. Can a customer buy several tickets for the same destination together or must they be bought one at a time?
2. Can customers cancel a request if a mistake has been made?
3. How should the system respond if an invalid card is input?
4. What happens if customers try to put their card in before selecting a destination (as they would in ATM machines)?
5. Must the user press the start button again if they wish to buy another ticket to a different destination?
6. Should the system only sell tickets between the station where the machine is situated and direct connections or should it include all possible destinations?

©Ian Sommerville 2014