Software Engineering 9

Solutions Manual

IAN SOMMERVILLE

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
January 2010

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.

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

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.