# **EXERCISES – Software Engineering (Ian Sommerville)**

## Chapter 1

1. **Explain why professional software that is developed for a customer is not simply the programs that have been developed and delivered.**Professional software includes not only the programs that have been developed and delivered, but also:
   1. Associated documentation, such as user or system documentation. User documentation should explain how to use the programs, and system documentation should go into more technical details about how the systems work together. The latter is usually targeted at software developers, and other technical roles.
   2. Associated libraries.
   3. Support websites.
   4. Configuration data that are necessary to make these programs work.

Indeed, professional software might include multiple separate programs and configuration files working together as a system. This is an important distinction between professional and amateur development: the latter does not assume maintainability, useability, nor dependability.

1. **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?**A generic software product is intended for the open market, whereas a custom software product has been designed for a specific customer / business case. The main distinction between generic software product development and custom software development, is that, for the latter, the developers must adhere to specifications that are determined by the customer. In contrast, for generic products, the organisation that develops the software controls the software specification, making it more flexible for developers: if they run into development problems, they can rethink what is to be developed.   
   In practice, this distinction might mean that users of generic software products must rely on the organisation that has developed the product. If the organisation did not follow good software engineering practices, for example, the product may be unreliable, or unsecure, putting the final users at higher risk. In addition, users might need to adapt their processes to fit the software rather than having software tailored to their processes. Furthermore, generic products might not perfectly match all users’ specific needs – with users normally having limited influence over product features compared to custom software clients. Of course, there may also be potential benefits, such as lower costs, more extensive testing due to larger user base, and potentially more regular updates.
2. **What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant.**Four important attributes that all professional software should have, are:
   1. **Acceptability** – software must be understandable, usable, and compatible with other systems users may utilise.
   2. **Dependability and security** – dependability includes a range of characteristics including reliability, security and safety. In the event of system failures, dependable software should not cause physical or economic damage. Malicious users should be prevented from exploiting the system by dependable software.
   3. **Efficiency** – software should not make wasteful use of system resources such as memory and processor cycles.
   4. **Maintainability** – software should be written in a way that would allow for the system to meet the changing needs of the business. Change is the only constant in software development.

Four other attributes that may sometimes be significant in professional software are:

1. **Compliance** – software must be designed to adhere to data privacy and retention regulations, such as the European Union’s General Data Protection Regulation (GDPR). For example, if under GDPR’s jurisdiction, software must process data lawfully, fairly, and in a transparent manner. Organisations developing software must clearly communicate how they collect and use personal data.
2. **Sustainability** – this attribute could be related to efficiency, but it goes further in making environmental sustainability as a priority. Organisations should take their software carbon footprint into account in the way it is designed, developed, and deployed and by rethinking some aspects of how the data centres that provide cloud-based services operate. For example, sustainable coding practices include optimising algorithms, reducing code complexity, and minimising the use of resources. Carbon-efficient software results from multiple factors – from selecting energy-efficient programming languages and platforms to “green” architecture design and DevOps. For example, interpreted languages consume 48x times more energy than compiled ones.
3. **Scalability** – this is the property of a system to handle a growing amount of work. In many cases it refers to the software’s ability to handle increased workloads while adding users and removing them with minimal cost impact.
4. **Testability** – software testability is the degree to which a software artifact (e.g. a software system, module, requirement, or design document) supports testing in each test context. If the testability of an artifact is high, then finding faults in the system (if any) by means of testing is easier.
5. **Apart from the challenges of heterogeneity, business and social change and trust and security, suggest other problems and challenges that software engineering is likely to face in the 21st century.**One of the greatest challenges software engineering faces in the 21st century is the Artificial Intelligence (AI) market hype. Over-reliance on AI tools by developers there’s a risk of homogenised solutions being more vulnerable to systematic exploitation putting cybersecurity at risk. In addition, this may erode fundamental programming skills and deep technical understanding, making debugging and testing harder. Finally, training these large AI models requires massive computational resources, creating tension between advancing AI capabilities and environmental sustainability.   
   However, this environmental impact is not specific to AI-based systems. Modern software systems, especially data centres, consume enormous amounts of electricity. Software engineers must design more energy-efficient systems to reduce carbon emissions. Another example is that constant software upgrades often drive hardware obsolescence, contributing to e-waste. Developing software that can run efficiently on older hardware could help address this issue, but it won’t solve it. As climate change accelerates, software engineering practices will need to incorporate sustainability principles throughout the development lifecycle.  
   Finally, the rapid pace of technological change creates constant demand for new skills the educational systems often struggle to provide. Communications between industry, the academic and public sectors need to be consolidated to develop programmes tailored at developing these skills.  
   Beyond the educational challenges, software engineering must also contend with the growing complexity of modern systems. As software continues to expand in scale and scope, managing ultra-large-scale systems with billions of lines of code across distributed environments poses unprecedented complexity challenges for testing, maintenance, and evolution. Additionally, emerging technologies such as quantum computing will require entirely new programming paradigms, security approaches, and ways of thinking about computation that current educational frameworks aren't prepared to address. Perhaps most concerning are the ethical dimensions of increasingly autonomous software systems. As software makes more decisions affecting human lives – from healthcare resource allocation to autonomous vehicles to criminal justice risk assessments – software engineers face complex ethical considerations regarding fairness, transparency, and accountability. These ethical challenges require not just technical solutions but interdisciplinary approaches that bring together computer scientists, ethicists, legal experts, and domain specialists to ensure software serves humanity's best interests.
6. **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.**
7. **Explain why the fundamental software engineering principles of process, dependability, requirements management and reuse are relevant to all types of software system.**
8. **Explain how the universal use of the web has changed software systems and software systems engineering.**
9. **Discuss whether professional engineers should be licensed in the same way as doctors or lawyers.**
10. **For each of the clauses in the ACM/IEEE Code of Ethics shown in Figure 1.4, propose an appropriate example that illustrates that clause.**
11. **To help counter terrorism, many countries are planning or have developed computer systems that track large numbers of their citizens and their actions. Clearly this has privacy implications. Discuss the ethics of working on the development of this type of system.**