ELEC-E8408 Embedded Systems Development

MODEL ANSWERS EXAM 7.4.2017

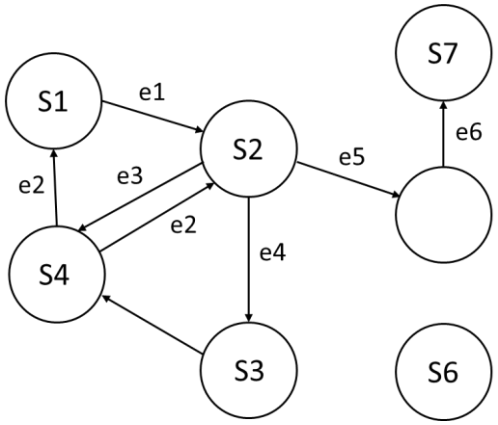
---

1. Are the following claims true (**T**) or false (**F**)? Every correct answer gives you +1 p, every incorrect –1 p, and an empty answer is worth 0 p. The minimum amount of total points is 0 p and maximum 6 p.

   a) Requirements elicitation involves gathering various requirements through a variety of techniques that may include stakeholder interviews, company-wide workshops, and prototyping. **T** (textbook p. 199)
   b) Software partitioning into multiple software units with low external coupling and high internal cohesion can be achieved through the principle of information hiding. **T** (textbook p. 284)
   c) A design pattern is a named problem–solution pair that can be applied in different contexts, with explicit advice on how to apply it in new situations. **T** (textbook p. 295)
   d) The KLOC metric takes into account the complexity of the software involved. **F** (textbook p. 420)
   e) In general, testing can detect the absence of errors in embedded real-time software. **F** (textbook p. 448)
   f) It is easier to develop platform-independent software code for multi-core platforms than for single-core ones. **F** (textbook p. 488)

2. Below is a Moore-type Finite State Machine (FSM) for the autonomous embedded control system of the *Aalto-1* satellite. The primary state sequence S2–S3–S4 is meant to be executed at least once in every two minutes.

   However, the FSM was drawn with a hurry and it contains six different fundamental defects. Identify each of them and explain why they should be considered as defects. (1 p for each defect; maximum 6 p) (textbook pp. 205–209)



There are the following six different defects:

1. The **initial state** is not defined at all.
2. Event **e2** triggers transitions from state **S4** to both **S1** and **S2**; while a transition from **S4** (or any state) can only go to a single state at a time (here either to **S1** or **S2**).
3. The state between **S2** and **S7** has no label.
4. The transition from **S3** to **S4** has no label.
5. The state **S6** is an isolated state without any entry or exit transition possibility (all states must be reachable).
6. The state **S7** is a terminal state, but in this case, there cannot be any terminal states, because "The primary state sequence **S2–S3–S4** is meant to be executed at least once in every two minutes." And this is a space application where no external assistance is available for service/maintenance.

3. Software engineering principles *Generality* (3 p) and *Anticipation of Change* (3 p) are seen advantageous for the following software qualities: *Interoperability*, *Maintainability*, and *Portability*.

   Why is there such a mapping between these principles and qualities? (textbook pp. 273–274, 278–280 and 281–282; discussed thoroughly during the lectures)

   Interoperability refers to the ability of the software to coexist and cooperate with other relevant software.

   Maintainability can be broken down into two contributing properties: evolvability and repairability.

   Portability of software is a measure how easily the software can be made to run in different environments.

   Software undergoes changes during its lifecycle. One approach is to anticipate where changes are likely to occur and make provisions for the changes during the design stage (for example, device drivers can be seen as a tool of **anticipation of change**). The principle of anticipation of change recognizes also the complexity of the learning process for both developers and their customers. And if a software component has several tasks rolled up into one package, it is likely that it will need to be split up when changes are made.

   While all these qualities, Interoperability, Maintainability and Portability, contain a significant "change" aspect, it is natural that preparation for changes – or Anticipation of Change – makes it easier to enhance such desired qualities.

   3 p for answers with relevant core arguments.

   A software engineer should consider what might be the underlying, hidden general problem that requires the solution. The general problem may not be as complex as the original problem, or there may be a ready-made solution library that is capable of satisfying the requirements. In addition, it is beneficial to design software in such a way that it is free from unnatural restrictions and limitations (for example, use configurable parameters instead of fixed constants). **Generality** is emphasized in both of these cases, as well as when creating interfaces between software units.

   Generality prepares the software for possible changes in the operating environment or use with somewhat different specifications, and makes it possible to reuse existing "general" software components. This advances the three desired software qualities.

   3 p for answers with relevant core arguments.

4. Describe three of the *values* listed in the Agile Manifesto (3 p) as well as three Agile Principles that you consider important for embedded software development (3 p).

Definition: Agile Manifesto (textbook p. 308)

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan

That is, while there is value on the items on the right, we value items on the left more.

Any combination of three of those four values is acceptable (1 p for each correct value).

Definition: Agile Principles (textbook p. 309)

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Any combination of three of those 12 principles is acceptable (1 p for each correct principle).

5. Explain the principles of *black-box* and *white-box* testing (3 p), and compare their advantages and disadvantages in software unit testing (3 p). (textbook pp. 449–450)

Black-box tests are data driven.

In black-box testing, only inputs and outputs of the code are considered; how the outputs are generated based on a particular set of inputs is totally ignored.

For each software unit, a number of test cases can be generated. This number depends on the number of inputs, the functionality of the software unit, and so forth.

Black-box testing techniques include: Exhaustive testing, Boundary-value testing, Random test-case generation, and Worst-case testing.

If a software unit fails to pass a unit-level test, then the detected error must be repaired, and all previous unit-level test cases are rerun to prevent the repair from causing other errors.

White-box tests are logic driven.

They are designed to exercise all paths in the code unit.

1.5 p + 1.5 p for correct black-box and white-box answers.

Black-box testing, being independent of the implementation of the software unit, can be applied to any number of units with the same functionality.

This technique does not provide any insight into the programmer's skills in implementing the software unit. Consequently, dead or unreachable code cannot be detected.

In addition, it may not test all of the flow paths in the software unit.

An important aspect of using black-box testing techniques is that clearly defined interfaces to the software units are required.

White-box tests exercise all flow paths of the code unit and provide also insight into the programming style.

White-box testing has the advantage that it can discover those code paths that cannot be executed. Such unreachable code is undesirable because it is likely a sign that the underlying logic is incorrect, because it wastes memory space, and since it might inadvertently be executed in the case of the corruption of the CPU's program counter.

1.5 p + 1.5 p for correct black-box and white-box answers.