# Home Exam – Software Engineering – D7032E – 2016

Teacher: Josef Hallberg, josef.hallberg@ltu.se, A3305

## Instructions

- The home exam is an individual examination.
- The home exam is to be handed in by **Friday October 28, at 17.00**, please upload your answers in Canvas (in the Assignment 6 hand-in area) as a compressed file (preferably one of following: rar, tar, zip), containing a pdf file with answers to the written questions and any diagrams/pictures you may wish to include, and your code. Place all files in a folder named as your username before compressing it (it's easier for me to keep the hand-ins apart when I decompress them). If for some reason you can not use Canvas (should only be one or two people) you can email the Home Exam to me. Note that you can NOT email a zip file since the mail filters remove these, so use another compression format. Use subject "**D7032E: Home Exam**" so your hand-in won't get lost (I will send a reply by email within a few days if I've received it).
- Every page should have your full **name** (such that a singular page can be matched to you) and each page should be numbered.
- It should contain ***original*** work. You are NOT allowed to copy information from the Internet and other sources directly. It also means that it is not allowed to cheat, in any interpretation of the word. You are allowed to discuss questions with class-mates but you must provide your own answers.
- Your external references for your work should be referenced in the hand in text. All external references should be complete and included in a separate section at the end of the hand in.
- The language can be Swedish or English.
- The text should be language wise correct and the examiner reserves the right to refuse to correct a hand-in that does not use a correct/readable language. Remember to spellcheck your document before you submit it.
- Write in running text (i.e. not just bullets) – but be short, concrete and to the point!
- Use a 12 point text size in a readable font.
- It's fine to draw pictures by hand and scanning them, or take a photo of a drawing and include the picture; however make sure that the quality is good enough for the picture to be clear.
- Judgment will be based on the following questions:
  - Is the answer complete? (Does it actually answer the question?)
  - Is the answer technically correct? (Is the answer feasible?)
  - Are selections and decisions motivated? (Is the answer based on facts?)
  - Are references correctly included where needed and correctly? (Not just loose facts?)

| Total points: 25 | |
|---|---|
| Grade | Required points |
| 5 | 22p |
| 4 | 18p |
| 3 | 13p |
| U (Fail) | 0-12p |

**Good luck! /Josef**

## Questions

**1. The Software Engineering context of Software Architecture** (3p, max 0.5 pages)
A large company with multiple distributed development teams is tasked with creating a software from scratch that is big enough to involve all the teams. Describe what software development model you would choose for such a task and what important architectural design decisions the software architect has to make to make sure the development runs smoothly despite the multiple distributed development teams. Also describe what kind of communication and/or documentation that you think should be used to support the software development model you have chosen.

**2. System Architecture** (4p, max 1 page excluding diagrams)
You are creating a calendar application like Google calendar. Give 4 examples of *design patterns* or *architectural patterns* that you think would be useful for the development of the application. For each example:

- Name the pattern
- Explain how, why (the advantage(s) of using it), and where it would be used and implemented.
- Optional: draw a diagram that illustrate how the pattern would be implemented

**3. Architecture design / Software maintenance** (2p, max 0.5 pages)
Discuss why we want low coupling and high cohesion when designing a software architecture. Motivate your answer, and give two examples of quality attributes in which low coupling and high cohesion are of extra importance.

**4. Testing** (1p, max 0.5 pages)
Explain why many companies write more code for tests than they do for the software itself. Also explain why it is important to have high test-coverage and why it may be difficult to achieve.

**5. Implementation** (2p, max 0.5 pages)
For each of the following tools/diagrams describe to what extent they are useful to the developer while implementing and testing the software, and how the developer would use them (motivate your answers):

- Use-case diagrams
- Functional specification
- Module/Class diagrams
- Stand-up meetings as part of an agile software development model approach

**6. Quality Attributes, Design Patterns, Documentation, Re-engineering, Testing** (13p)
A newbie programmer has created a simple monopoly game called LTUMonopoly. The newbie is quite happy with the look and feel of the game, but wants it to be more flexible and extensible. The newbie wants the monopoly game to be customizable so that it can be tailored to specific towns, workplaces, etc. Please help the newbie by re-engineering the code and create a better design and code, which is easier to understand. There is no documentation other than the comments made inside the code, but the basic idea of the game is provided inside of the `paintBoardGame` function.
The source code is provided in the `LTUMonopoly.java` file (compile with: `javac LTUMonopoly.java` run with: `java LTUMonopoly [Player 1-4 initials]`).

The newbie wishes the software architecture design was better made so that it would be easy to customize/extend the monopoly board, game mechanics, and chance cards for us by other organizations (like a Luleå monopoly, a CompanyA monopoly, MyFriends monopoly, etc.). Your task is to completely re-engineer the code and pay close attention to good software architecture design and coding best practices (keeping in mind the quality attributes: coupling, cohesion, sufficiency, completeness, primitiveness - http://atomicobject.com/pages/OO+Quality). You should design your architecture for *Modifiability* (see slides for lecture 6), *Customizability* (make the software easy to customize for different organizations) and *Comprehensability* (make the code easy to read and understand), and utilize code reuse and design-patterns where appropriate. You are of course allowed to add additional classes and change as much of the code as you wish/need, as long as the game plays and looks roughly the same. Follow the re-engineering principles presented lecture 9.

Add unit-tests, which verifies that the game runs correctly (it should result in a pass or fail output, preferably color coded), where appropriate. The syntax for running the unit-test should also be clearly documented. Note that the implementation of the unit-tests should not interfere with the rest of the design.

Examination criteria - Your code will be judged on the following criteria:

- Whether it is easy for a developer to customize the board, game mechanics, and chance cards.
- How easy it is for a developer to understand your code by giving your files/code a quick glance.
- To what extent the coding best-practices have been utilized.
- Whether you have paid attention to the quality metrics when you re-engineered the code.
- Whether you have used appropriate design-patterns in your design.
- Whether you have used appropriate variable/function names.
- To what extent you have managed to clean up the messy code.
- Whether program uses appropriate error handling and error reporting.
- Whether the code is appropriately documented.
- Whether you have created the appropriate unit-tests.

*If you are unfamiliar with Java you may re-engineer the code in another object-oriented programming language. However, instructions need to be provided on how to compile and run the code on either a Windows or MacOS machine (including where to find the appropriate compiler if not provided by the OS by default).*