Examination
Software Analysis and Design
DIT185

Software Engineering and Management
Chalmers | University of Gothenburg

Wednesday August 23, 2023

Time	08:30-12:30
Location	Lindholmen
Examiner	Birgit Penzenstadler
Responsible teacher	Daniel Strüber (mobile: 0760475434)
Total number of pages	**7** (including this page)
Teacher visits exam hall:	At circa 09:30 and at circa 11:30
Exam (4.5 HEC)	Max score: 100 pts
Grade limits (4.5 HEC)	3: at least 50 pts
	4: at least 70 pts
	5: at least 85 pts

ALLOWED AID:

- English dictionary
- **NOT ALLOWED:** Anything else not explicitly mentioned above (including additional books, other notes, previous exams, or any form of electronic device: dictionaries, agendas, computers, mobile phones, etc.)

PLEASE OBSERVE THE FOLLOWING:

- Start each question on a new paper;
- Sort your answers in order (by question and sub-task) before handing them in;
- Write your student code on each page and put the number of the question on **every** paper;
- In several tasks, you will be asked to create a UML diagram. If any aspects of your created diagrams are ambiguous or potentially unclear without further explanation, please provide a textual description that explains these aspects.
- We set up this exam in such way that you can realistically complete it in four hours. In particular, the diagrams are generally smaller than the ones from the assignments.
- This exam is composed by eight exam tasks, some of them with further sub-tasks. Tasks 1-6 are based on the same case, a submission system for student papers (described on page 2).
- Points are denoted for each task and sub-task. The point distribution can give you an indicator of how much time to spend on each task and sub-task.

System description for Tasks 1-6:

We consider a *submission system* for managing student papers for assignment tasks (think Canvas). The submission system has the following requirements:

- Every course in the system has lecturers assigned to it. This is done by one of the course administrators, who is also a lecturer. As part of a course, lecturers may create tasks and assess papers submitted by students. Therefore, the lecturers award points and give feedback.
- The course administrator defines which lecturer assesses which papers. At the end of the course, the course administrator also arranges for certificates to be issued. A student's grade is calculated based on the total number of points achieved for the submissions handed in. Students can take courses and upload papers.
- All users—students and lecturers—can manage their user data, view the courses and the tasks set for the courses (provided the respective user is involved in the course), and view submitted papers as well as grade points. However, students can only view their own papers and the related grades. Lecturers can only view the papers assigned to them and the grades they have given. The course administrator has access rights for all data.
- A course is created and deleted by an administrator.
- When a course is created, at least one administrator must be assigned to it. Further course administrators can be assigned at a later point in time or assignments to courses can be deleted. The administrator can also delete whole courses. Information about users and administrators is automatically transferred from another system. Therefore, functions that allow the creation of user data are not necessary.
- All of the system functions can only be used by persons who are logged in.

Usage scenario for Task 4 (sequence diagram):

A student uploads the solved assignment paper to the submission system. The system informs the course administrator that a new assignment paper has been submitted and confirms to the student that the paper has been successfully received. Via the submission system, the course administrator assigns a lecturer to the paper. Once the system has informed the lecturer that a paper has been assigned, the lecturer assesses the paper. To do this, the lecturer downloads the paper from the submission system, grades the paper, and enters the grade in the system. Then the student is informed that the uploaded paper has been graded.

This interaction between the system and its users takes place not just once, but for every assignment that has to be completed for a course. Once all of the assignments have been processed, the course administrator arranges for the certificate to be issued. The submission system then informs the student of the final grade.

Task 1: Requirement modeling (12 pt.)

a) Create a use case diagram for the submission system. Your use case diagram should include all use cases that are mentioned in the system description, and only use cases that are actually mentioned in it. Make sure that your use-case diagram is well-structured: It should have a system boundary, at least two actors outside that boundary, relations from the actors to all use cases (and labeled extensions where appropriate). Where appropriate, align use cases in a hierarchy (<<extend>>, <<include>> relationships). (8 pt.)

b) Provide correct description (according to the Cockburn template) for one central use case from your use case diagram. Include at least one extension and one sub-variation. You can find the Cockburn template on the last page of this examination sheet. (4 pt.)

Deliverables:

- Task 1a: Use case diagram
- Task 1a: Textual description (only when needed – see hint on cover page)
- Task 1b: Use case description

Task 2: Domain modelling (12 pt.)

a) Create a domain model, in the form of a class diagram, for the submission system. Recall that a domain model depicts the concepts from the real-world domain that need representation in the system. Develop your domain model in a systematic way by applying guidelines and heuristics, such as those from the lectures. Create classes with important associations (including multiplicities, where applicable) and attributes. Create at least one generalization (inheritance relationship). (8 pt.)

b) Provide a textual description that explains how you developed the domain model, including specific explanations of how you used guidelines and heuristics. If any aspect of your domain model is not self-explanatory, explain it here. (4 pt.)

Deliverables:

- Task 2a) Domain model (class diagram)
- Task 2b) Textual description of how you developed domain model (half a page)

Task 3: Behavior modelling: state machine diagram (12 pt.)

a) Create a design model, in the form of a state machine diagram, that models the different states of submissions (a.k.a. student papers). Include at least four states that submissions can be in, and the possible transitions between the states. (8 pt.)

b) Illustrate your state chart diagram with two object diagrams. Choose a transition from your state machine diagram, and create two object diagrams showing snapshots of the system directly *before* and *after* the transition. It's okay to show a small excerpt of the system – you can focus on those parts that are affected by the transition. (4 pt.)

Deliverables:

- Task 3a) Design model (state machine diagram)
- Task 3b) Two object diagrams, name of transition

Task 4: Behavior modelling: sequence diagram (12 pt.)

Create a sequence diagram that models the usage scenario as described on page 2. Use a white-box modeling approach: that is, your model should include at least two interacting objects of the submission system. You can assume that there is a design model that contains the relevant classes for that (you don't need to draw the design model).

Deliverables:

- Task 4: Sequence diagram, modeled in white-box way

Task 5: Architecture modelling (12 pt.)

a) Create a component diagram for the submission system. For this, choose an architectural style and create a component diagram for the application based on it. For this task, assume that the submission system should be supported by different user interfaces, e.g., through several tablet apps and through a website. Model at least three interacting components together with their interactions (either through interfaces or dependencies). (8 pt.)

b) Explain the design rationale for your component diagram: explain the reasons for choosing the architectural style, and make references to relevant design principles. (4 pt.)

Deliverables:

- Task 5a) Architecture model (component diagram)
- Task 5b) Textual description of rationale

Task 6: Consistency (10 pt.)

We consider a collection of consistency rules for class diagrams and sequence diagrams, which are listed on the last page of this examination sheet.

1. Based on the consistency rules, create two examples of inconsistencies between class diagrams and sequence diagrams. For each example, draw a small class diagram and a small sequence diagram, and name the consistency rule that is violated in the example. That's four diagrams in total. The two examples must violate different consistency rules (not the same rule for both examples). (5 pt.)

2. Consider your class diagram from Task 2 and your sequence diagram from Task 4. The class diagram is probably not consistent with the sequence diagram yet, because it is a domain model. However, it's possible to make the class diagram consistent, by

making a few changes to it, turning it into a design model.
Which changes are required? Please give concrete steps with details (e.g., names of concrete elements added) to explain how the class diagram can be made consistent with the sequence diagram. (5 pt.)

Deliverables:

- Task 6a: Two examples, including a pair of models and a violated consistency rule
- Task 6b: Description of steps for establishing consistency.

Task 7: Applying design patterns to three cases (15 pt.)

Below, you will be presented with three small example cases. For each of them, provide the design based on an appropriate design pattern. Use a different design pattern for each case. For each case, name the design pattern and create a class diagram showing how the implemented design pattern looks like for the case. (5 points per case)

Case 1: Weather monitoring system. Suppose you are developing a weather monitoring system that needs to send notifications to multiple weather stations when a weather event occurs. The weather stations may be located in different parts of the region and may have different ways of displaying the weather data. The system should not be tightly coupled to the weather stations, meaning that the system should not need to know about the specific details of each weather station. Additionally, weather stations may need to subscribe and unsubscribe from the notifications dynamically without causing any disruption to the rest of the system.
Please use a design pattern for handling of whether events.

Case 2: Leaderboard feature in game. Suppose you are developing a game with a leaderboard feature where players can submit their scores and see their rankings. Since the game will be used by many people at the same time, it's important that the score data is maintained in a coordinated way: A single object should be responsible for handling all score data management.
Please use a design pattern for making sure that there is indeed a single object handling all the score management.

Case 3: Clothing online store. Suppose you are developing an e-commerce system that sells clothing items. Each clothing item has a base price, but customers can select additional features such as custom embroidery or monogramming, which can increase the final price. Instead of creating separate classes for each combination of features, we want a flexible system that allows customers to dynamically add the features they want.
Please use a design pattern that supports the flexible combination of additional features for clothing items.

Deliverables:

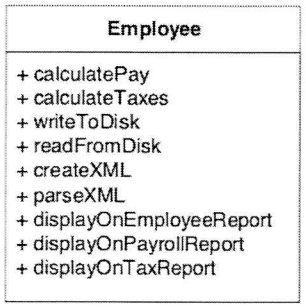
- Task 7: For each case, a class diagram and the name of design pattern

Task 8: Improving software quality via design models (15 pt.)

a) Critically discuss the software quality of the class diagram below. Explicitly address at least two design principles. (5 pt.)

b) Create an improved version of the class diagram. (5 pt.)

c) Discuss the rationale for your improved class diagram: why does it lead to an improvement? Explicitly address the design principles again that you previously addressed in task 8a).




```
public class Employee {
  public double calculatePay();
  public double calculateTaxes();
  public void writeToDisk();
  public void readFromDisk();
  public String createXML();
  public void parseXML(String xml);
  public void displayOnEmployeeReport(
                        PrintStream stream);
  public void displayOnPayrollReport(
                        PrintStream stream);
  public void displayOnTaxReport(
                        PrintStream stream);
}
```

Explanations: Methods `calculatePay` and `calculateTaxes` calculate the salary and the taxes of the employee. Method `writeToDisk` saves the information about an employee to a file in the file system, `readFromDisk` reads the information about an employee from a previously saved file in the file system. The methods `createXML` and `parseXML` create a String representation of an employee's information (in XML format) and read an employee's information from the same type of String representation, respectively. The methods `displayOnEmployeeReport`, `displayOnPayrollReport`, and `displayOnTexReport` determine how the employee information is represented in different types of reports: the created representation will be added to the provided `PrintStream`.

Deliverables:

- Task 8a: Textual discussion of software quality (one paragraph).
- Task 8b: Class diagram
- Task 8c: Textual discussion of rationale (one paragraph)

Appendix

For Task 2: Cockburn Template (relevant excerpt, to be filled out by you)

Use case name:	<the name should be the goal as a short active verb phrase>
Goal in context:	<a longer statement of the goal>
Primary Actor:	<a role name for the primary actor, or description>
Precondition:	<what we expect is already the state of the world>
Success End Condition:	<the state of the world upon successful completion>
Failed End Condition:	<the state of the world if goal abandoned>
Trigger:	<the action upon the system that starts the use case, may be time event>
Main Success Scenario:	<put here the steps of the scenario from trigger to goal delivery, and any cleanup after> <step #> <action description>
Extensions:	<put here the extensions, one at a time, each referring to the step of the main scenario> <step altered> <condition>: <action or sub use case> <step altered> <condition>: <action or sub use case>
Sub-Variations:	<put here the variations that will cause eventual bifurcation in the scenario> <step or variation #> <list of variations> <step or variation #> <list of variations>

For Task 6: Consistency Rules for Class Diagrams vs. Sequence Diagrams

1. The type of a lifeline (type of the connectable element of the lifeline) in a sequence diagram must not be an interface nor an abstract class.
2. If a message in a sequence diagram is referring to an operation, that operation must not be abstract.
3. If a message in a sequence diagram refers to an operation, through the signature of the message, then that operation must belong, as per the class diagram, to the class that types the target lifeline of the message.
4. Interactions between objects in a sequence diagram, specifically the numbers of types of interacting objects, must comply with the multiplicity restrictions specified by the class diagram (e.g., association end multiplicities).
5. In order for objects to exchange messages in a sequence diagram, the sending object must have a handle to the receiving object as specified in the class diagram. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.
6. The behavioral semantics of a composition or aggregation association in the class diagram must be inferred in sequence diagrams. For instance, in a whole-part (composition) relation, the part should not outlive the whole.
7. Each public method in a class diagram triggers a message in at least one sequence diagram.
8. Each class in the class diagram must be instantiated in a sequence diagram.
9. No operation can be used in a message of a sequence diagram if this breaks the visibility rules of the class diagram (public, protected, private).