Homogenization of classes

Lecture # 29
16 Nov

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Software Design and Analysis
CS-3002

Today's Outline

- Checking the Model
- Why Homogenize?
- Combining Classes
- Splitting Classes
- Eliminating Classes
- Consistency Checking
- Scenario Walk-Through
- Event Tracing
- Documentation Review

Homogenization of the system

- The word homogenize means to blend into a smooth mixture, to make homogeneous.
- In parallel design process, several stimuli with the same purpose or meaning are defined by several designers. These stimuli should be consolidated to obtain as few stimuli as possible. It is called homogenization.
- Homogenize - to change (something) so that its parts are the same or similar.

Why Homogenize?

- As more use cases and scenarios are developed it is necessary to make the model homogeneous. This is especially true if multiple teams are working on different parts of the model.
- Since use cases and scenarios deal with the written word, people may use different words to mean the same thing or they may interpret words differently. This is just the natural maturation of the model during the project life cycle.

Why Homogenize?

- Homogenization does not happen at one point in the life cycle—it must be on-going.
- Projects that wait until the end to synch up information developed by multiple groups of people are doomed to failure.
- The most successful projects are made up of teams that have constant communication mechanisms employed. Communication may be as simple as a phone call or as formal as a scheduled meeting—it all depends on the project and the nature of the need to talk.
- The only thing that matters is the fact that the teams are not working in isolation.

Elements of Homogenization

- Combining Classes
- Splitting Classes
- Eliminating Classes
- Consistency Checking
- Scenario Walk-Through
- Event Tracing
- Documentation Review

Combining Classes

- If different teams are working on different scenarios, a class may be called by different names. The name conflicts must be resolved.
- This is accomplished mainly through model walk-throughs. i.e.
  - Examine each class along with its definition.
  - Also examine the attributes and operations defined for the classes, and look for the use of synonyms.
  - Once you determine that two classes are doing the same thing, choose the class with the name that is closest to the language used by the customers.

Combining Classes

- Pay careful attention to the control classes created for the system.
- Initially, one control class is allocated per use case. This might be overkill—control classes with similar behavior may be combined.
- Examine the sequencing logic in the control classes for the system. If it is very similar, the control classes may be combined into one control class.
- E. g In the Course Registration System there is a control class for the Maintain Course Information use case and one for the Create Course Catalog use case. Each control class gets information from a boundary class and deals with course information. It is possible that these two control classes could be combined since they have similar behavior and access similar information.

Splitting Classes

- Classes should be examined to determine if they are following the golden rule of OO, which states that a class should do one thing and do it really well.
- They should be cohesive;
- Example :
- A StudentInformation class contains:
  - Info about student Actor
  - And info about the courses that student has successfully completed.
- This is better modeled as two classes—StudentInformation and Transcript, with an association between them.

Splitting Classes

- Often, what appears to be only an attribute ends up having structure and behavior unto itself and should be spilt off into its own class.
- For example, we'll look at Departments in the university. Each Course is sponsored by a Department. Initially, this information was modeled as an attribute of the Course class.
- Further analysis showed that it was necessary to capture the number of students taking classes in each department, the number of professors that teach department courses, and the number of courses offered by each department.
- Hence, a Department class was created. The initial attribute of Department for a Course was replaced with an association between Course and Department.

Eliminating Classes

- A class may be eliminated altogether from the model.
- **This happens when:**
  - The class does not have any structure or behavior
  - The class does not participate in any use cases

- Guideline , examine control classes.
- Lack of sequencing responsibility may lead to the deletion of the control class. This is especially true if the control class is only a passthrough— that is, the control class receives information from a boundary class and immediately passes it to an entity class without the need for sequencing logic.

Eliminating Classes

- In the Course Registration System, initially we would create a control class for the Select Courses to Teach use case.
- This use case provides the capability for professors to state what course offerings they will teach in a given semester.
- There is no sequencing logic needed for the control class—the professor enters the information on the GUI screen and the Professor is added to the selected offering.
- Here is a case where the control class for the use case could be eliminated.

Consistency Checking

- Consistency checking is needed since the static view of the system, as shown in class diagrams, and the dynamic view of the system, as shown in use case diagrams and interaction diagrams, are under development in parallel.
- Because both views are under development concurrently they must be cross-checked to ensure that different assumptions or decisions are not being made in different views.

Consistency Checking

- Consistency checking does not occur during a separate phase or a single step of the analysis process. It should be integrated throughout the life cycle of the system under development.

- Consistency checking is best accomplished by forming a small team (five to six people at most) to do the work.

- The team should be composed of a cross-section of personnel— analysts and designers, customers or customer representatives, domain experts, and test personnel

Scenario Walk-Through

- A primary method of consistency checking is to walk through the high-risk scenarios as represented by a sequence or collaboration diagram.
- Since each message represents behavior of the receiving class, verify that each message is captured as an operation on the class diagram.
- Verify that two interacting objects have a pathway for communication via either an association or an aggregation.
- Especially check for reflexive relationships that may be needed since these relationships are easy to miss during analysis. Reflexive relationships are needed when multiple objects of the same class interact during a scenario.

Scenario Walk-Through

- For each class represented on the class diagram, make sure the class participates in at least one scenario. For each operation listed for a class, verify that either the operation is used in at least one scenario or it is needed for completeness.

- Finally, verify that each object included in a sequence or collaboration diagram belongs to a class on the class diagram.

Event Tracing

- **For every message shown in a sequence or collaboration diagram,**
  - verify that an operation on the sending class is responsible for sending the event and an operation on the receiving class expects the event and handles it.
  - Verify that there is an association or aggregation on the class diagram between the sending and receiving classes.
  - Add the relationship to the class diagram if it is missing.
  - Finally, if a state transition diagram for the class exists, verify that the event is represented on the diagram for the receiving class.
  - This is needed because the diagram shows all the events that a class may receive.

Documentation Review

- Each class should be documented!
- Check for uniqueness of class names and review all definitions for completeness.
- Ensure that all attributes and operations have a complete definition.
- Finally, check that all standards, format specifications, and content rules established for the project have been followed.

That is all