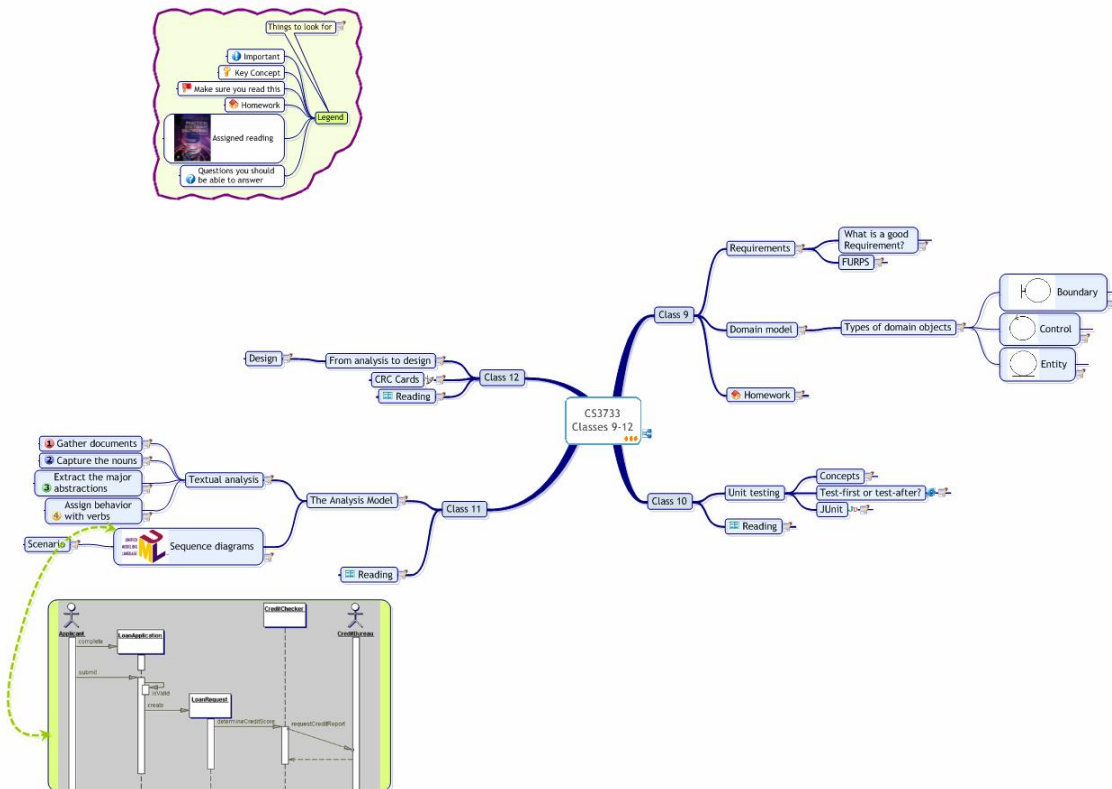


CS3733 Classes 9-12



See document: [CS3733 SoftwareEngineering.mmap](#)

Class 9

Requirements

So far we've considered functional requirements and how they can be represented by use cases. There are other types of requirements that you need when specifying a system.

What is a good Requirement?

A good requirement is one that has the following characteristics:

- **Specificity.** The requirement must be specific to one aspect of the system. Although requirements may be stated at different levels, they should address one facet of the system only.
- **Testability.** The requirement must be testable. That is, you need to be able to determine that the requirement has been satisfied in the product you deliver.
- **Unambiguity.** The requirement must be unambiguous. While formal methods are not easy to read (and, for some people, write), they are ideal for specifying systems because of their well-defined, unambiguous semantics.

FURPS

FURPS is an acronym that is used to describe the types of requirements. Each letter stands for one type of requirement:

- **Functionality:** what the system must *do*. Use cases are used for specifying functional requirements.
- **Usability:** how easy must it be to use the system. These requirements are difficult to specify since it is so easy to simply quote "easy to use" as the requirement. This is not a good requirement.
- **Reliability:** what is the acceptable level of service that the system must provide? Does it need to be available 24 hours a day, 7 days a week? What is the maximum time the system can be taken down?
- **Performance:** the acceptable response time, process load, and so on.
- **Supportability (Scalability):** how easy is it to understand, maintain, and extend the system? How good is the design and architecture?

Domain model

The domain model is a conceptual model of the problem domain. It is an abstraction of the requirements, specified in terms of a set of objects that describes the system under development *in terms that someone who is a domain expert understands*.

This means that the domain model uses the same nouns and verbs as someone who works with the system. These concepts form a model of the "real world" as seen by the people who inhabit it.

The domain model captures those types of objects that are most important system under construction, *in the context of the system*.

Types of domain objects

There are several types of objects that you might identify for your domain model. They fall into three broad categories as described in the following subsections. However, don't forget to think about the underlying abstract concepts, as they might provide additional objects to consider.

Boundary

Boundary objects are those that are used by the system to interact with the outside world. These objects might represent I/O devices, or some communication connection to an external system. The accompanying icon shows how these might look when a UML diagram represents the <<boundary>> stereotype as an icon.

Control

Control objects usually represent a set of activities, or a process, that performs a related set of actions. These objects control, or manage the flow of actions and data flow. Sometimes you might have a control object that represents the control of a system, like a database manager.

Objects with the stereotype <<control>> may be shown with the accompanying icon.

Entity

Entity objects represent persistent (data) objects that the system works with. They usually represent the physical items in the problem domain, such as invoice, bill, warehouse, and so on.

Entity objects have the <<entity>> stereotype and may appear with the accompanying icon in a UML diagram.

Homework

Do exercise 1 on page 233 of the text. Also, take one of the use cases you identify for the exercise and write the textual description of it. Include the name, brief description, basic flow of events and subflows.

Class 10

Unit testing

Concepts

Unit testing is a fairly simple concept. When you produce some unit of work, you test it. From a quality viewpoint, you want to test throughout the software lifecycle. Errors cost in terms of rework, customer dissatisfaction, and so on. The longer an error lives in your system, the more expensive it is to remove it and its effects.

Unit testing is the first level of testing. It is usually done by the developer. As work progresses, units are combined to form larger units, called components, that will also be tested. But, just like constructing a building on a unstable foundation, you don't want to try to integrate and test units that are, themselves, not tested.

Test-first or test-after?

See document: [4929.html](#)

A popular technique today is *Test-first Programming (TFP)*. TFP says that you should write the tests for your code before you write the code itself. There are many reasons why TFP proponents believe that this is a better practice than writing the tests after you code. See the article on the attached link for more information about TFP.

You may write your unit tests after you have written your code. In this case you need to be careful that you don't design the test to make your code work. You want tests to reflect requirements, and your code to reflect the requirements as well.

For this class, you have the option of writing your tests first or after you write the code. However, the important things are:

- test early
- test often
- don't continue until *all* tests pass

JUnit

See document: [index.htm](#)

We will use JUnit to write tests for this class. JUnit is a simple test framework that supports both test-first and test-after approaches. It is integrated into the Eclipse IDE. See the JUnit home page by following the associated link for more information.

Reading

Chapter 8

Class 11

The Analysis Model

The analysis model is the domain model discussed in class 9. In this section we will look at techniques for identifying the domain objects, their properties and behavior.

Textual analysis

Textual analysis is one of the most straight forward ways of identifying domain objects. This section describes a procedure for performing textual analysis to arrive at the set of objects in your domain model.

Gather documents

You will, by this time have several documents that were either provided to you, or that you created. In this case, we use the term document very loosely. A document is any artifact that you have that describes the problem in some form.

Capture the nouns

The nouns in the documents are potential domain objects. Don't worry at this point if you have duplicates, just create a list of them. If there is some confusion about what the noun represents, then you need to provide a short description of it.

Extract the major abstractions

The domain model is not meant to be a collection of all of the possible objects that exist in the domain. Rather, it is a description of the most important concepts that you need to understand in order to build a solution to the problem. There are three steps in this activity:

- Group duplicates. It is common to have duplicate items. These are usually aliases for the same concept.
- Revise names as needed. This is especially useful if you've grouped several names together.
- Select the most important abstractions. You don't want all of objects you've identified. There will probably be many more than you need to describe the problem domain. Select only those that represent major concepts in the domain.

Assign behavior with verbs

The next step is to determine the behavior of the objects and how they interact. In order to do this, use the verbs that are used in conjunction with the objects from the previous step to determine the behavior necessary for the objects to do their job.

Sequence diagrams

A very effective method for capturing how a system behaves is through a sequence diagram in UML. The sequence diagram shows a sequence of actions that occur between objects. Usually you will create a sequence diagram for major scenarios of the use cases.

By creating and reviewing sequence diagrams, you are able to discuss and reason about the behavior of the existing system and how the new system should work.

Read the article on sequence diagrams at: <http://www.developer.com/design/article.php/3080941>.

Scenario

A scenario is *one path through a use case*, usually with specific values assigned. A scenario may cross many flows of events in a use case. It is always a description of a path that starts from the beginning of the use case and ends at some endpoint of the use case (either a normal end, or some exception).

Reading

Section 9.1

Class 12

From analysis to design

The purpose of analysis is to understand the problem in enough detail that you can begin to design a solution. As with all parts of your work, this will develop iteratively and incrementally.

You will add detail to your analysis model and it will evolve into your design model. The design model describes the architecture, structure, and behavior of the system you build.

Design

What do we mean by design? Design is the part of the process where we shape the system under construction. We define, in detail, the major classes and assign responsibilities to them. We apply good design patterns to produce an architecture that is resilient and capable of satisfying the requirements as stated, yet flexible enough to be expanded for future requirements.

Design is not just about producing the blueprints, but it also consists of creating some of the code that illustrates that the blueprints describe something that can be built and works.

CRC Cards

See document: [paper.html](#)

One of the simplest techniques for doing an initial design of a system is with CRC cards. CRC stands for Class- Responsibility-Collaboration. They provide a way for a team to quickly converge on an agreed-upon design for a system. See the linked article for an introduction to CRC cards.

Reading

Section 9.2.

<http://www-106.ibm.com/developerworks/rational/library/4929.html>