Introduction
The goal of these pages is to give a general overview of how you will develop your programming project along the next months. They do not mean to

• Explain thoroughly the concepts and techniques employed; this will be done in the theory sessions and in other documents you will have available.
• Detail the documents and software you have to prepare for each submission; this is done in the document "Course Rules" and it will also be explained along the course.

Here, it will be reviewed:

• The phases of the project,
• How they relate to the subject’s submissions,
• What is accomplished at each phase,
• What techniques are applied in each one,
• What knowledge is required.

The phases and the submissions
Your project will go through the following phases:


Obviously, many of these phases overlap in time. In particular, by "Documentation" we mean mainly the writing of the user manuals. However, all the phases need to be documented (both the work done and the decisions taken).

The correspondence between these stages and the three subject submissions are stated next:

• First submission: Specification and analysis
• Second submission: Design and start of coding and testing
• Third submission: End of coding and testing; documentation.

First submission
We will specify what our system will do by means of three documents:

• Detailed definition
  Textual description, as precise as possible, of what the system must accomplish, containing no ambiguities and clearly stating all the undefined points of the definition originally provided to you.

• Use case diagram
  In outline, a use case is a functionality of the system accessible from outside. For each use case, the following info must be stated:
  • Actors involved (in our case, probable the types of user of the system)
- Who starts the use case
- What is the typical dialogue between actors and the system, the event flow.
- What alternative event flows can take place (errors or unusual situations).

The use case diagram includes all the actors of your system as well as in what use cases each one is involved.

We will use UML (Unified Modelling Language) terminology for the use cases. We will use Rational Rose Modeller\(^1\) tool for drawing the diagrams and a conventional text processor for their specification (it might have been also done with RR but it generates extremely rambling listings).

- **Non-functional requirements** of the system: Those features of the program which are not linked to its functionalities such as hardware requirements, performance, security, user-friendliness…

At the Analysis stage we will start to deal with the object orientation paradigm. The main result of this phase is:

- **Domain classes diagram**
  In this diagram we will identify the objects that form the domain, those that are used to describe the problem; we find features shared by the objects and do abstraction of any differences among them, this way we group them into classes; we identify attributes which are common to the objects of a class; and we state some of the relationships between classes, "associations" and "aggregations".
  The class diagram is also expressed in UML using Rational Rose Modeller. The descriptions of the elements of the diagram can be introduced by jeans of the templates attached to the diagram but it is not compulsory, a word processor can be used for its specification.

**Knowledge required for this submission:**
- Being able to organize a document, write clearly and tidily, and without linguistic misspellings, for the natural language description.
- Being able to understand UML most simple use case diagrams and classes diagrams.
  (Watch out: UML is a very very extensive terminology. In this subject, we will only explain a very small part of it. You will see it in a more systematic way at the Software Engineering subjects).
- The first concepts of object orientation: object, class, attribute, associations and aggregations.
- Use the (small) corresponding part of Rational Rose Modeller tool.

---

**Second submission**

**New classes turn up**

We start from the classes diagram, which states what objects are involved in the problem, and from the use case diagram, which states what the program must allow to do with these objects (… these are the functionalities!).

---

\(^1\) Or alternatively either ArgoUML or any other similar tool.
In the projects we propose, most functionalities are used by the user to create, look up, update, relate or delete domain objects.

In order to design these functionalities, we will use a three layer architecture, which will be explained during the course. In this design pattern, the following classes of objects are involved in each functionality:

- One or more domain objects.
- One view and a presentation controller, objects responsible for presenting these objects to the user.
- Domain controllers: objects that keep the synchronisation between the domain and the view, that is, between the domain objects and how the user sees them.

On the other hand, in this phase we realise that objects can exist at the main memory... while the program is running! Usually the objects will have to be persistent: they must keep on existing between one execution of the program and the following one, and this means that we have to save them to disk.

We therefore need disk manager objects in order to implement persistence. If your application is connected to a data base, the disk managers will be the ones to deal with it. Since you are not necessarily supposed to know about data bases, you will use files in this subject.

Then, three new classes of objects appear, as well as the domain ones we already had: views, controllers and disk managers. These classes do not belong to the domain because they do not appear when we describe the problem (specification), but when we decide how to solve it (design).

Sharing with your cluster

The other two groups in your cluster must solve a problem that may initially seem quite different from yours. Therefore, you will probably have different classes. But for sure there are some classes that have some common features. In fact, probably some of the classes of your own project have common features between each other.

You will use the inheritance, genericity and polymorphism mechanisms of object orientation in order to not to have to program the same thing twice. You will put in common both your classes and the ones of the other groups and you will identify the generic and abstract classes that capture these shared features. Many of the specific classes of your final system will derive from these classes. You will agree on what classes (both generic and specific ones) will be shared by two or three of the groups of the cluster, and each one of them will be implemented by one group only. All the shared classes must be implemented for the second submission, although the other groups have two more weeks to accept them (or request corrections, in case of serious errors).

A coordination effort must be done in the cluster so that:

- You identify all that is worth to be shared, so that you need to implement less.
- The classes that are programmed by another group are the ones your group really needs. Specify together the shared classes, before programming them! This specification will have to be submitted a week before the second submission, and it must be detailed enough so that any two implementations meeting the specification are interchangeable.

Domain classes

Finally, the domain classes are implemented for this submission.

Take into account that some domain classes can be shared (in case the very same class appears in any of the other project definitions) and maybe another group of the cluster is assigned to implement it. However, most of the classes of your domain will be specific of your project definition and you will have to implement them yourselves.
Knowledge required for this submission:

- The basics of three layer architecture
- Java language
- Java compiler and its basic libraries
- From OO paradigm: inheritance, genericity and related concepts
- Program testing techniques.

Remember:
One of the main goals of the subject is to learn to write reusable software. That’s why we demand you to share classes with other groups. For the same reason, we will ask you to appropriately document all the classes you program, so that they can be easily reused.

Third submission
From this moment on, with both the domain classes and the shared classes implemented and working, you just have to implement use cases one after the other. This means to write the specific views, controllers and disk managers for the use case, and link them to the classes of your domain. Usually these views, controllers and disk managers inherit from generic classes that have already been programmed.

The final effort:
- Coding and testing of the use cases.
- Integration of all the use cases (although it is advisable to integrate them incrementally as they are implemented).
- Integration testing.
- Writing of the manuals.
- Preparing the submission and the interview with the tutor (it takes time).

Remember:
Each class must be implemented by only one person. This is requested not only in order to be able to evaluate your work individually. It is good to seek advice for the more delicate points with the team mates, but the strategy "We do all the classes a little bit altogether" will only make you spend many more hours than necessary.

Knowledge required for this submission:
- Same ones as for the second submission.
- Keep calm when the final submission day is closer and you still have to do this, and this, and also that…