

Introduction

Group MASTER, academic course 2007-08 Q1

The professor: Alicia Ageno (email: ageno@lsi.upc.edu)

The classes: a theory session and a practical assignment. In fact, we try to explain all the theory as soon as possible, to leave all the final sessions to practical assignment.

Two recommendations: be quiet (you can go in and out whenever you want) and be participatory, which means you can interrupt me whenever you want (sometimes it's good to ask a question at the right moment) it is good for you to loose fear to seem ridicule and feel shy uselessly.

The subject: the first I have to recommend you is a previous reading of the teaching guide, if you have not yet done it. You will find all the recommended bibliography there.

The subject's URL is: <http://www-assig.fib.upc.edu/~prop>. The person in charge of the subject is Alicia Ageno.

It is a practical subject where you will have to *wise up*, because there is a certain quantity of knowledge that you will have to learn by yourselves and you will be asked to apply your own criterion.

In the syllabus the subject is placed after PRED and in parallel with subjects in the field of Software Engineering (SE). The purpose is to collect all the acquired knowledges from the programming subjects (including ADA and BD that can be taken in parallel) to do a practical assignment a lot bigger (a project) and to be also an introduction to SE. [In the previous syllabus it was placed between EDA and ES:E.] Those students that have taken an ES course will see the rudimentary use of all the related knowledge (you should get adapted to the level given in this subject in order to be fair with your classmates).

We are aware that the number of credits assigned to this subject is maybe a bit too low according to the work that will be done. That is why it is so important that you learn to make good use of the time, organize yourselves the better, so you will have to schedule a good give out of tasks.

We are also aware that if you have strictly followed the syllabus you will know nothing or very little about file management and databases, but you will have to use them. That is why we will be tolerant with this part of the project. (The students with a higher level in this matter will also need to adapt their level to the one of their mates.)

Essentially the purpose of the subject is to build a project, which results in a program that should end up working without errors (neither aborting nor producing wrong results).

You will have to work in groups. To do the project, 3 statements will be published; each of them must be implemented by a group of 3 students (in some cases ± 1). These groups of three are grouped into three of three (a total of 9 students, in the worst case ± 3), this "super group" will be named "cluster". As a whole, the *cluster* will

work on the three different statements, that is, the first group will implement the project with the first statement, the second group, the one with the second one, the third group, the one with the third one.

You have the chance to state your desire to form a group or *cluster* with the classmates you know. The subject will usually respect your states to form a complete *cluster* in this case you can also say which statement will be done by each group. We understand that in these petitions you have to respect that the groups will be formed by three students. You can also propose incomplete *clusters* (one or two groups and also with incomplete groups of 1 or 2 students, etc.). However these cases make things more complicated so there will not be a guarantee of respecting the petitions.

You should send your requests as soon as possible, always *before hour 24 h of the next September 19th, 2007*. For your requests you should fill the form that you will find on the subject's website:

<http://www-assig.fib.upc.edu/~prop/formulari.txt>
and send it to ageno@lsi.upc.edu.

The entire *cluster* will have the same tutor. The subject has a forum on the web. You can use it in order to try and find students to complete your groups. The list of *clusters* will be published on the subject's site.

Along the course you will have to do 3 submissions that will be explained in the laboratory classes, the first two will grade the group in general and the last one will also give an individual grade for each member of the group. That is why you have to separate the source code files programmed by each student. In the last submission all members of the group have to be present. The final grade is calculated as follows: 10% 1st submission + 25% 2nd + 65% 3rd.

Only the first submission has to be done totally in paper. The rest will be done in diskette (a few specific documents will also be accepted in paper). It is necessary to know and respect the subject rules for these submissions.

The tutor can accept your work or not, in case he/she doesn't accept it he/she might give you the instructions to complete it in order to be accepted. The fact of having to correct a delayed submission will affect your grade negatively. One time it can mean your submission to be punctuated over 6 points, a second, over 4 points.

The subject considers the project as a real project and not as a simple academia assignment, that is to say, as if you had to sell it when it's finished. That is why, in addition to data structures and algorithms, formal aspects of presentation are also important (specially in the first submission), the ease of comprehension of your work (for the tutor to correct it, he/she has to understand everything easily and clearly), the ecologic matters and other that we will discuss have an important impact in a real project (interface ergonomics, documentation, etc). This includes aspects that one may regard as secondary as a good writing, binding or orthography.

Engineering: why do we call it *software engineering* when we speak of building programs and we don't speak of manufacturing?

In an industrial process, for example to build a fridge or a car, the first thing to do is to define the model and then manufacture it. The first phase consists in defin-

ing how the product must be to the last detail and to build a first model (a fridge or car that is subject to laboratory and real tests). After it is necessary to design and start the production chain, nowadays normally an automatic process, in order to manufacture the product that has to be sold in the real market. Next, there will be the real product manufacturing, and finally it will have to be packed and distributed in order to arrive to the customers.

That first phase in which both the product and the design of the production chain are defined is product *engineering*, once the production chain is started the manufacturing is done. Notice that when we build programs there isn't practically a manufacturing, someone may think that the manufacturing phase corresponds to the production of the distribution support (CD, diskettes, FTP, DVD, etc.) and the execution of the installing process, but it is rather the fact of packing the product and distribute it. So we say that in software construction there is only engineering, and not manufacturing.

The lifecycle of a computer application: we speak of a product's lifecycle when defining the phases and stages it takes, since we first start to think and define it until it is not used or useful anymore. In the case of a computer application it will include the moment in which the need or convenience to make it is stated until it is uninstalled or the last copy is left unused.

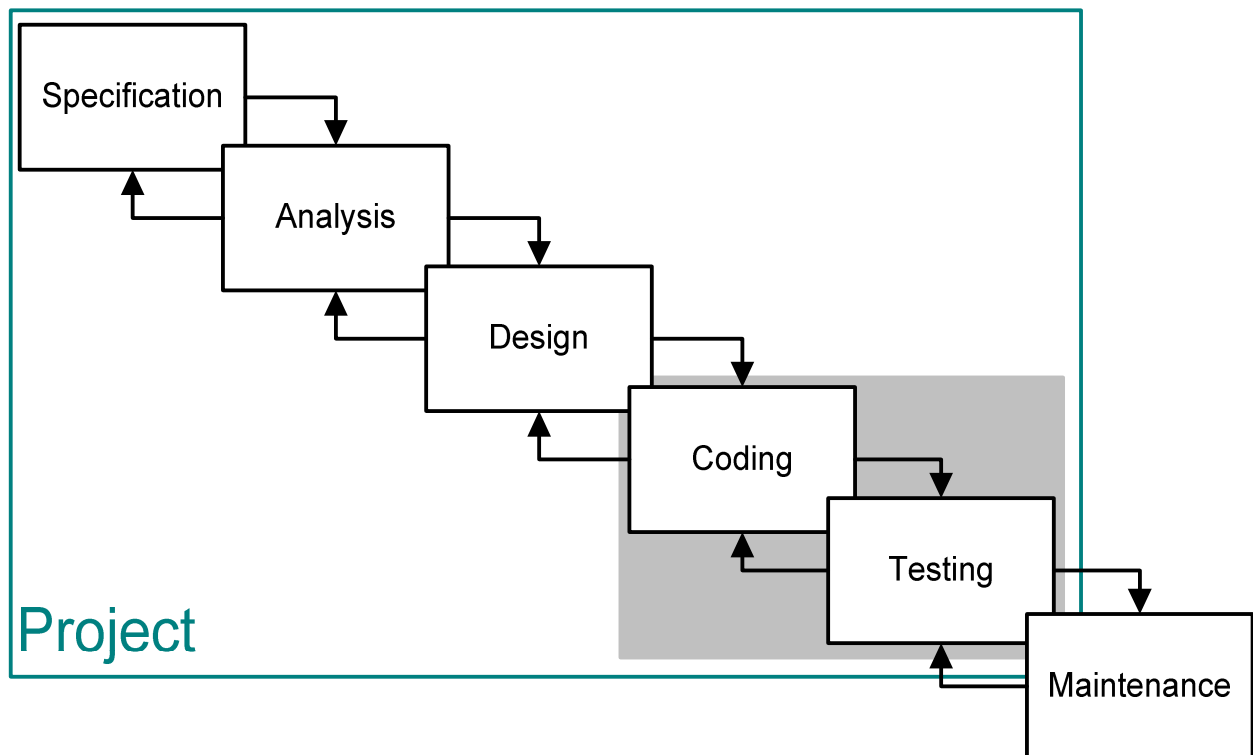
Normally, the lifecycle is part of the development methodology to be used. Nowadays we can find almost as many lifecycles as methodologies, but we could group them into a reduced number of families. As it is not a subject's goal to show the different methodologies and lifecycles panorama we will only speak about the *classic lifecycle*, although it is not very used nowadays because its ideas are not quite realistic today. An example is the absence of backtracking, since the constant evolving of business makes them being continuously adapting their computer softwares. That is so important, that it is normal that before finishing an application some of the requirements and requisites will have changed or new ones will have appeared.

However, this model is correct to do this project, because the system to obtain must follow some concrete requisites and is totally stable.

This lifecycle is also named as cascade lifecycle, because it is drawn as a descendant stair and because it was initially considered that a stage started when the previous was finished. It was not thought that you could go back, from a stage to the following a highly concrete documentation was passed that was the basis of the work to be done but it could not allow to work with it or reuse it, that is to say the different stages were disjoint.

Nowadays all the process is considered as iterative, it is always possible that from one stage a revision of the previous is needed, so the backtracking is allowed (ascendant way). In addition the idea of *seamless transition* (which means the documentation of a stage is used in the following one as a direct tool to work on), is introduced. So the stages are not disjoint, a certain overlapping is found.

The following is this lifecycle schema:



1) The specification defines what the program has to do; in fact, it is part of a bigger study that will include the cost (economic and temporal), the suitability and the definition or software specification. This specification must include the functionalities definition, of what has to be able to do. As we will see, this stage is delicate because the success of the project relies on it, at least a big part of it.

2) The analysis is the first stage that is purely technical; it studies and defines the system functionalities. In this stage the concrete technological aspects of implementation (as the programming language, file management system or databases, operating system, etc) are not yet important but the functional aspects as the data structures and treatments.

3) The design is about giving an informatics solution, so we must define processes (algorithms), files or databases (the management system to use), etc. Here, what will be important are the adopted technological solutions, the architectonic model of the programs, the equipment to be used, the operating systems, programming languages...

4) The coding is to write the source code in the programming language to be used.

5) Testing is about seeing whether the programs behave correctly, do what they have to (satisfy the specification), never abort, etc. In fact, this phase is usually done in parallel with the previous one, as we will see, normally an iteration of coding and testing is done.

6) The maintenance stage starts in the moment the program is considered to be finished, and continues as long as the software continues to be used. There are four types:

1.- Corrective: correct problems, defects of origin or construction.

- 2.- Adaptive: adapt to the changes that could happen (for example a new law) and affect it.
- 3.- Perfectionist: to make it better. Add more functionalities or improve the existing ones.
- 4.- Preventive: solve problems before they happen (as the famous 2000 effect).

In fact, the Project finishes with the program's installation, which is when maintenance begins. Sometimes a maintenance operation can start a whole new project.

The names given to the lifecycle stages you may find at the bibliography are not always the ones used in this document, you can find others (synonyms), but the ones presented in this document will be the ones used all over the subject.

We will also give concrete indications about the necessary adaptations to work with object orientation.

Project development: once we know this lifecycle we can take a look at how the subject works reviewing the following document: "General view of a programming project" (<http://www-assig.fib.upc.edu/~prop/VisioGeneral-EN.pdf>), which you will do or already have done in the first laboratory session.