Software Development Process

André Fonseca

Affiliation not available

July 12, 2017

1 Software Process

- Waterfall
- Evolutionary prototyping
- Rational Unified Process (RUP)
- Agile

2 Software Phases

1. Requirements Engineering: Talk to the customer, to the stake holders, whoever we are building the software for. Understand what kind of system we need to build.
2. Design: Use the previous information to design the high-level structure, that can become more detailed, of our software system.
3. Implementation: Write code that implements the design.
4. Verification and validation: We need to make sure that the code behaves as intended.
5. Maintenence: Several activities like add new functionality, eliminating bugs from the code or responsing to problems that were reported from the field after we released the software.

2.1 Requirements Engineering

The process of establishing the needs of stakeholders that are to be solved by software.

Iterative process that loops over each phase multiple times in the same order:

1. Elicitation: Collection of requirements from stake holders and other sources. Can be done in multiple ways.
2. Analysis: Involves the study and a deeper understadint of the collective requirements.
3. Specification: Collective requirements are suitable represented, organized and saved so that they can be shared. Can be done in multiple ways.
4. Validation: Make sure that they're complete, consistent, no redundant and so on.
5. Management: Accounts for changes to requirements during the lifetime of the project.

2.2 Design

The phase where software requirements are analyzed in order to produce a description of the internal structure and organization of the system. This description will serve as the basis for the construction of the actual system. A series of design activities that go from a high-level architectural design, to a low-level view.:

1. Architectual design
2. Abstract specification
3. Interface design
4. Componente design
5. Data structure
6. Algorithm design

2.3 Implementation

Realizing the design of the system that we just created and create an actual software system.

Four principles:

- Reduction of complexity: Aims to builds software that is easir to understand and use.
- Anticipation of diversity: Takes into account that software construction might change in various way over time. Software evolves and in many cases it evolves in unexpected ways.
- Structuring for validation or design for testability: Build software that it is easily testable.
- Use of (external) standards: Coding standards or naming standards to be valid in the domain.

2.4 Verification and Validation

Aims to check that the software system meets it specification andfulfulls its intende purpose. More precisely, we can look at verification and validation independently.

- Validation: is the activity that awnsers the question: Did we build the system that the customer wants? That will make the customer happy.
- Verification: Did we build the system right? So given a description of the system that is the one that we derived from the customer through the collection of requirements, then design and so on, did we build a system that actually implements the specification that we defined? Verification can be performed at multiples levels:
  - Unit level: Test individually units work as expected.
  - Integration level: Test interaction between the different units. The different modules talk to each other the right way.
  - System level: Test the system as a whole. Make sure that all the system, all the different pieces of the system work together in the right way.

2.5 Maintenance

When the software is released and operational many things can change, like the environment, libraries, etc, or new features requests or users might find problems with the software.

Maintenance activities:

- Corrective maintenance: To eliminate problems with the code
- Perfective maintenance: Accomodate new feature requests and in some cases just to improve the software (make it more effecient)
- Adaptive maintenance: To take care of the environment changes.

Then a new version is released and the cycle will continue. During maintenance everytime you modify your application you have to regression test the application - the activity of retesting software after it has been modified to make sure that the changes work as aspected and do not introduce any unforseen effect.

3 Software Process Model (Software lifecycle)

How we put these activities together to develop software? A model of what should happend at the start and finish of the software development process.

3.1 Waterfall

The project progresses to an orderly sequence of steps. At the end of each phase there will be a a review to determine whether the project is ready to advance to the next phase. Performs well for softer products in which there is a stable product definition, the domain is well known and the technologies involved are well understood.

Advantages: In these cases it helps to finds errors early on.

Disadvantages: Not flexible. It is difficult to specify requirements at the beginning of a project, and this kind of flexibility is far from ideal when dealing with a project in which requirements change, the developers are not domain experts or the technology used are new and evolving.

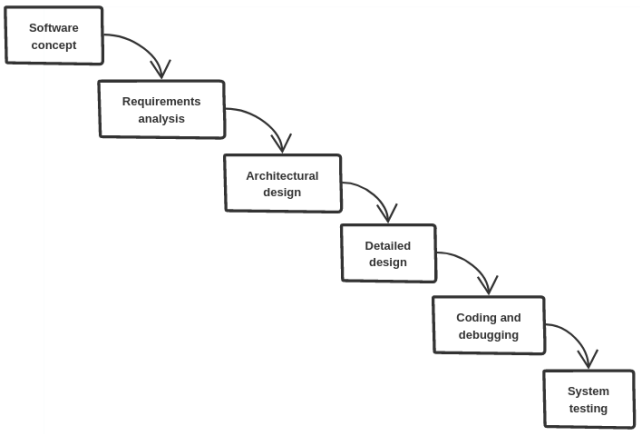

Figure 1: Waterfall model

3.2 Spiral model

Incremental risk-oriented lifecycle model that has four main phases:

1. Determine objectives
2. Identifiy and resolve risks
3. Development and tests
4. Plan the next iteration

Advantages: Extensive risk analysis does reduce the changes of the project to fail. Functionality can be added at a later phase. Software is produced early with prototypes. Early feedback from the customer about what we produced.

Disadvantages: The risk of analysis requires a highly specific expertise. The whole success of the process is highly dependent on risk analysis. Risk analysis

has to be done right. It is way more complex than other models and can be costly.

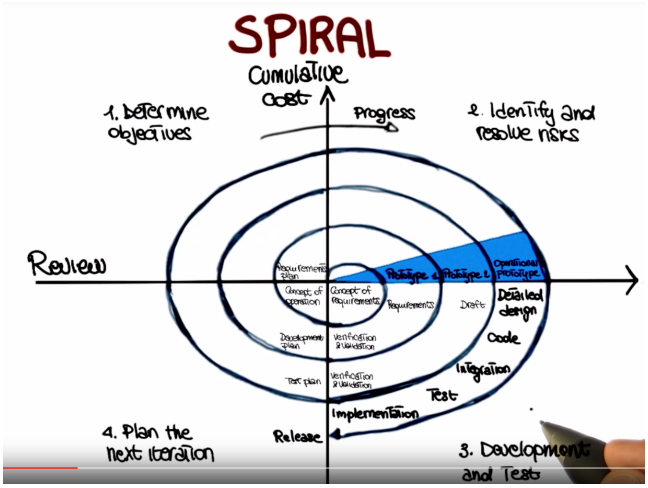

Figure 2: Spiral model

3.3 Evolutionary prototyping

Works in four main phases:

1. Initial concept
2. Design and implement initial prototype
3. Refine prototype until acceptable
4. Complete and release prototype

The system is continually refined and rebuilt. So it is an ideal process when not all requirements are well understood, which is a very common situation. The developers start by developing the parts of the system that they understand, instead of working on developing a whole system, including parts that might not be very clear at that stage. The partial system is then shown to the customer and the customer feedback is used to drive the next iteration, in which either changes are made to the current features or new features are added.

Advantages: Immediate feedback as soon as they produce a prototype and show it to the costumer.

Disadvantages: Difficult to plan in advance how long the development is going to take, because we don't know how many iterations will be needed.

3.4 Rational Unified Process (RUP)

Based on UML. Works in an iterative way, which means it that it performs different iterations, and at each iterations it performs four phases

3.5 Agile

Highly iterative and incremental development. Test driven development (TDD):

1. Red: Test cases that encode our requirements and for which we haven't written code yet and therefore, they will fail, obviously.
2. Green: Write enough code to make test cases pass. Satisfy the requirements.
3. Refactor: Modify to make it more readable, more maintainable. Modify to improve the design of the code.

And continue to iterate among these phases.
General project info for Software Development Processes: https://www.udacity.com/wiki/sdp/proj