1. **LIFE CYCLE Model**

**It is a series of steps while software product is developed and maintained**

**Type of Life cycle model:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Steps** | **Pros** | **Cons** |
| Waterfall | Requirement->Analysis-> Design->Implementation | * Disciplined Approach * Document   Driven | Delivered product may not meet the clients need |
| Rapid-Prototyping | Rapid-prototype->Analysis->design->Implementation->posted delivery maintenance->retirement | * Ensures that the delivered product meets the client needs | Not yet proven beyond all doubt |
| Evolution | Same as waterfall but maintenance is different | * Closely model real world software production * Better communication | Integration |
| Spiral | Same as rapid prototyping but each step contains risk analysis | * Very safe | Expensive  More time  Only for large scale |

**5 GENERATION OF PROGRAMMING LANGUAGE**

**First Generation**: Machine Code

**Second Generation**: Assembly Language

**Third Generation**: Procedural Programming Language (functional) :C, PASCAL, Fort , Cobal

**Fourth Generation**: Object Oriented: Java, C#, Scripting : Python, Matlab, Javascript,

Embeded: SQL

**Fifth Generatio**n : AI, Turing Machine ( still dream

How to reduce mutual misunderstanding:

3 types of methods are there to describe a system:

* Informal --

Example: Natural language

**Pros: Under stable**

**Cons: Full of ambiguity, not accurate**

* Semiformal --

Example: ER diagram, Used Case, Class Diagram ***(Illustrate)***

**Pros: Strict, used in real world, avoid ambiguity**

**Cons: Hard to understand**

* Formal --

Example: Petri Net, Finite State Machine, and Axiom ***(Illustrate)***

**Pros: Very easy to transfer into code, Not ambiguity, Prove correctness**

**Cons: Full of math/ logic, very Hard to understand**

*OOA -- Object Oriented Analysis* (THEY ARE INDEPENDENT)

In the specification phase, you want to analyze your system based on the concept of the object. You treat the system as an object.

*OOD -- Object Oriented Design*

**2 types of Design:**

* **High level (Modules: coupling, cohesion)**

Concept is based on modules.

Run view of a system.

High-level design this is the only concept are used.

* **Low level: Detailed design**

*OOP – Object Oriented Programming*

Support class, OO support!

Reuse (if programming language is OO)

It is possible to use OOP to design a system!!!!

**Coupling: 2modules. How close or lose they are. Loose is better you get to work independently. Easy to maintain and test.**

Level of Coupling:

1. Content : Private Variable
2. Common: Global Variable
3. Control: One module controlling the flow of another
4. Stamp: Not all information useful
5. Data: All information useful
6. No Direct Relation:

**Cohesion: Measure of how strongly related or focused the responsibilities of a single module are. It should be very close.**

Level of cohesion:

1. Coincidental : (Worst) Nothing in common
2. Logical : Something in common in many cases
3. Temporal : a --------------------> a1
4. Procedural : cause and effect a1 must a2 take csc221 and then take csc322
5. Communication: share some data
6. Sequential: parts of module are grouped because the output from one part is the input to another part like an assembly line.
7. Functional: Best: When parts of module are grouped because they all contribute to a well-defined task of the module.

Low level (abstract and data structure)

Algorithm + Data Structure

Design Real World Problem:

You come up with the right algorithm

Complexity must be O(long n)

Math or Programming

Recursive formula and improve it

Design the system based on classes

Functional Programming: It is based on LIST, and everywhere.

Finite State Machine/ Petri Net (Small Petri Net)

**Finite State Machine or Petri Net**

**Design a Low level**

**Real Programming**

ER Diagram – It shows how to design database

Collaboration Diagram / Sequence Diagram – (DFD got replaced!) – Class Diagram

**2) PLANNING**

**How to Estimate Cost**: Several leading Methods to estimate cost & pros & cons

|  |  |  |  |
| --- | --- | --- | --- |
| method | Des | Pros | Cons |
| Time | Human years/hour | Simple to decide to total | Cheating |
| Funtionality | hire experts to decide total | Hard to cheat, | Hard to come up with the fare price, and who will decide the price/cost |
| Number of lines | Calculated by per lines | Simple to decide total, | Cheating by adding extra lines and lower quality |
| Analog and expreince | Experts from ur team to decide cost | Free and honest | Competent |
| Expert | Hire another expert consultatnt | Easy | Expensive and cheat |
| By weighted sum | Different price for different parts | Easy | Error in estimating cost |

**Team Format**: Very Important part of software engineering

Democratic: No leader (everyone equally important or unimportant

Pros:w

Cons: To much communication, no one will listen to other, no authority

Chief: Good in everything, and split the work among everyone else.

Pros: Communication channel is very small.

Cons:

Real Industry:

2 heads all the time:

Technical – Very good in programming, senior analysis

Manager – Very good in social skill

**How to choose good and right people --**

**Which programming language to use –**

**3) DESIGN**

25 horse Race

8 bottol poison

Skip List

Recursive – iterative

Space/ Time 🡪 efficiency

Don’t use too much space… To much time

Consider: Hashing and sorting

How to Sort: & Pros & Cons

How to Hash: & Pros & Cons

**4) CODING**

Any language (based on your design)

**Knuth Letter (goto):**

* Comment needed
* Efficiency : time /space
* understandability

**5) TESTING**

* **Black box ( in this case you check system by portioning, typical cases and boundary cases ->> this is functional testing which is based on input and output)**
* **White box testing (not popular, hard to test whole system and impossible to implement)**

Coding + Test your own code

Few concepts is testing:

* **Alpha version: internal testing, version that passes the internal testing**
* **Beta version: Google everything is in beta version, I am not responsible**
* **Deliver version:**

**Double Blindness Testing: HOW TO DESIGN IT 🡪 testing should be objective and effective (randomization)**

Debug Vs. Testing:

* **Debug is not testing.. Done by the programmer, belong to the coding phase**
* **Testing: Done by different people, no association.**

Samples: Why samples are important!!

Randomization + Uniform:

**Testing:**

**Static Testing: Read your code, check and test based on specs**

**Dynamic Testing: Run and Excel system**

**Write Code (statistical Testing-> Relaibilty which is based on number of errors, use time , nature of errors and seriousness of errors)**

* **Verification: System Meet Specification, Most important part of measure**

**VS**

* **Validation: Verification + More measure (User friendly + Better GUI)**

DOUBLE BLINDNESS: **Cause and effect association testing**

* **Experimentation: active testing**
* **Observation: Passing testing à run system and observe (association)**
* **Stress testing à testing should be independent and blindness**

**6) MAINTENCE + EVOLUTION**

**Evolution:** maintence + reengineering

* + - Corrective is required
    - Adaptive is optional
    - Perfective is optional

**Re-engineering:** (Forward) software engineering , still have to use life cycle model

**Reverse engineering**: Executable->source code ->design ->specs