Introduction to Software Engineering

Program: An implementation of an algorithm in a programming language

Software: Collection of programs + Software Documentation (technical) + User Documentation + configuration files.

S/w Documentation: explains the structure of the system + UML/ER diagrams + Function call-tree diagrams

User Documentation: How to use the software (user guide)
Websites for updates/recent downloads
How to activate the product /Licence Release

Configuration files: files related to Linker, shell scripts (auto run)...

Engineering: How to make 'things' work / how to develop products / how to convert ideas into products

Software Engineering: Concerned with developing software products, how to sell the products, how to provide customer support/service

Types of software products

Generic Product: SRS (Software Requirement Specification) is generated by the developer itself. SRS is converted into a product and it is sold to any customer

Examples: OS, Databases, Word-processors, Drawing tools, Explorer, Editors

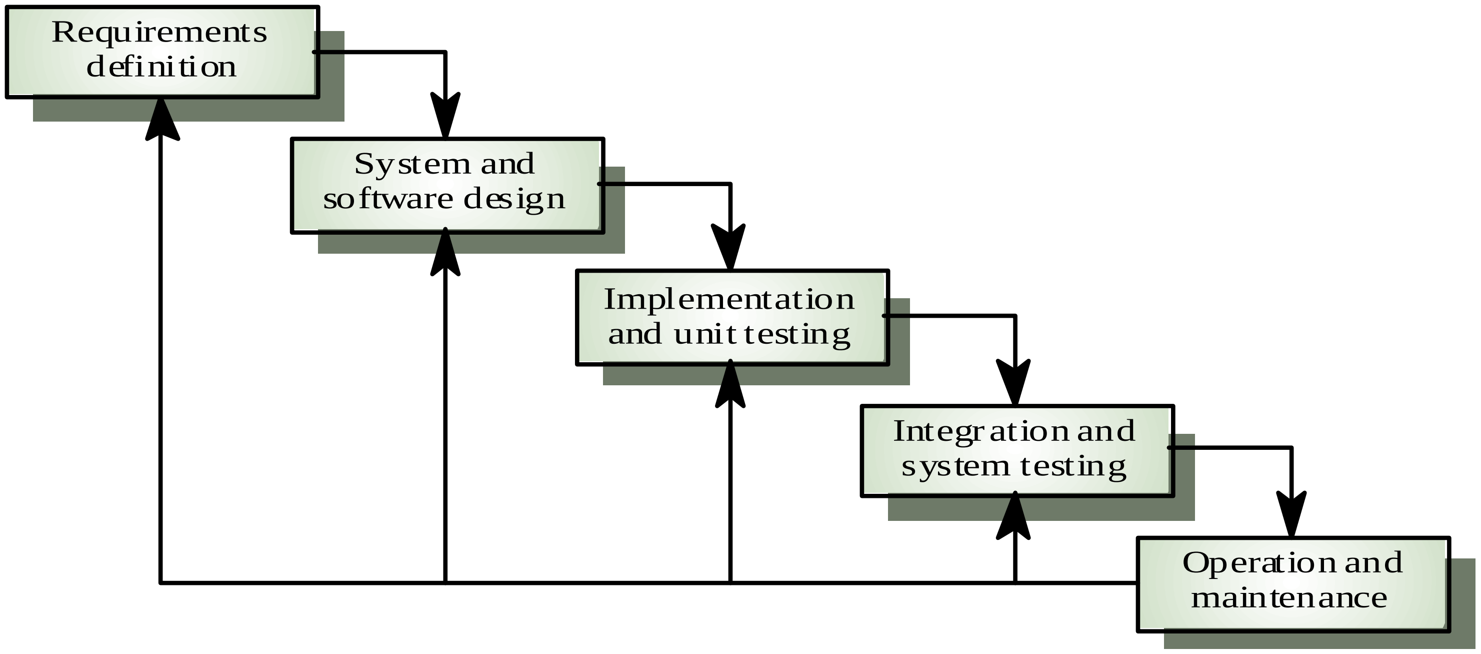
Customized Product: SRS is specified by a particular customer. A software contractor develops the software especially for that customer.

Examples: IRCTC, Inventory System, Tax-calculator

How about this idea? Develop a generic system and customize it to specific application.

ERP: Enterprise Resource Planning (SAP)
Koha: Open Source Library Management Software (Customize: Koha-IIITDM)
Tally-ERP- Accounts Software

Waterfall Process Model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Waterfall: cascading of one phase with another
(also known as software life cycle or SDLC (s/w development LC))

Requirements Defn: Services to be provided, constraints and goals are established in consultation with USER

Software Design: Design partitions the requirements into Hardware and Software.
Design gives overall system architecture.

Implementation: Design is realized as a set of programs or program units.

Unit testing: Verifying whether each unit meets its specifications.

Integration: Program units are integrated and tested as a complete system to ensure that SRS is met. After this stage, product is delivered to the customer.

Maintenance: This is the longest life-cycle phase. It involves correcting errors which were not discovered during unit testing. Also, new program units will be introduced to meet 'dynamic' SRS.

Developing a Simplified Computer
Product Name: SIMPUTER

SIMPUTER

Basic Arithmetic Operations	Linear Algebra Operations	Scientific Operations	Sorting and Searching
Addition()	Matrix_Addition()	Sin()	Linear_Search()
Subtraction()	Matrix_Multiply()	Cos()	Binary_Search()
Multiplication()	Determinant()	Random()	Ternary_Search()
Division()	Inverse()	Binomial()	Merge_sort()
Factorial()	Eigenvalue()	Poisson()	Insertion_sort()
Power()		Std_deviation()	Selection_sort()

DECLARATION (Separating INTERFACE from Implementation)

```
// BASIC.h

Class
Basic_Operation
{
        private:
        Public:
                Addition();
                Multiply();
                Factorial();
                        .
};
```

```
// LINEAR.h

Class
Linear_Algebra
{
   private:
   Public:

Matrix_Addition();
    Matrix_Multiply();
    Determinant();
};
```

```
// SCIENTIFIC.h

Class Scientific_OP
{
   private:
   Public:
       Sin();
       Cos();
       Random();
                .
};
```

```
//
SORT_SEARCH.h

Class Sort_Search
{
   private:
   Public:
       Bin_search();
       Merge_sort();
     Insertion_sort();
                .
};
```

DEFINITION (IMPLEMENTATION)

```
// BASIC.cpp
#include"BASIC.h"

Basic_Operation:: Addition()
{

}
Basic_Operation:: Factorial()
{

}
```

1. Where will you include main() - only once in the main file

2. Does each .cpp contain main() - NO

3. can Sin() in Scientific.h invoke Power(), Factorial() in Basic.h - Yes

4. How to compile .h and .cpp files

BASIC.h | Linear_Algebra.h | Scientific.h | Sorting_Searching.h

BASIC.cpp | Linear_Algebra.cpp | Scientific.cpp | Sorting_Searching.cpp

SIMPUTER.h

SIMPUTER.cpp

```
#include<iostream.h>
#include"Basic.h"
#include"Linear_algebra.h"
#include"Scientific.h"
#include"Sorting_Searching.h"

Class Simpute
{
        private:
        Public:
                select_operation()
                accept_input()
};
```

```
#include"Simputer.h"

Main ()

// create objects

Simpute simpute_object;

simpute_object.accept_input()
{

}

Basic_Operation Basic_object;
Basic_object.addition()
```

Modes of Compilation when multiple files involved

Mode: 1 (user defined executable file instead of a.out)

```
g++ simputer.cpp basic.cpp Linear_algebra.cpp Scientific_operations.cpp
Sorting_Searching.cpp   -o  FINAL_PRODUCT
```

To see the output: ./FINAL_PRODUCT

Mode: 2 (generate object files explictly followed by executable)

```
g++ -c simputer.cpp
g++ -c basic.cpp
g++ -c Linear_algebra.cpp
g++ -c Scientific_operations.cpp
g++ -c Sorting_Searching.cpp

g++ simputer.o basic.o   Linear_algebra.o Scientific_operations.o
Sorting_Searching.o -o FINAL_PRODUCT
```

To see the output: ./FINAL_PRODUCT

Mode: 3   Using **Makefile**

Module Integration using MAKEFILE

```
all: FINAL_PRODUCT

FINAL_PRODUCT:  simputer.o basic.o Linear_algebra.o Scientific_operations.o
Sorting_Searching.o
	g++ simputer.o basic.o   Linear_algebra.o Scientific_operations.o Sorting_Searching.o -o
		FINAL_PRODUCT

simputer.o:simputer.cpp
	g++ -c simputer.cpp

basic.o: basic.cpp
	g++ -c basic.cpp

Linear_Algebra.o: Linear_Algebra.cpp
	g++ -c Linear_Algebra.cpp

Scientific_operations.o: Scientific_operations.cpp
	g++ -c Scientific_operations.cpp


Sorting_Searching.o: Sorting_Searching.cpp
	g++ -c Sorting_Searching.cpp
```

Black labels : Target
Blue labels: Dependencies
Red labels: Command

To create executable
Run

Make all

To see the product
./FINAL_PRODUCT

Product Delivery

SIMPUTER (Main folder)

src **obj** **include** **doc** simputer

Src folder: all .cpp files + makefile
Obj folder: all .o files
include folder: all .h files
Doc folder: Readme file, User doc, technical doc
Simputer: executable file

Revisiting Makefile

```
All: Final_Product

Final_Product : object/Basic.o object/LinearAlgebra.o object/Scientific.o object/Sort_sear.o
                object/simputer.o

    g++ -o Final_Product object/Basic.o object/LinearAlgebra.o object/Scientific.o
                        object/Sort_sear.o  object/simputer.o


object/simputer.o : src/simputer.cpp

    g++ -c src/simputer.cpp && mv simputer.o object/simputer.o

object/Basic.o : src/Basic.cpp

    g++ -c src/Basic.cpp && mv Basic.o object/Basic.o
```