Basics of Software Engineering

● Outline:
  – Overview
  – Software Development Life Cycle
  – Project management
  – Requirements
  – Analysis and design
  – Implementation
  – Testing
  – Maintenance
Overview
Overview

- **Software Engineering:**
  - **Software:**
    - Programming code (program code + libraries + interfaces + …)
    - Collection of executables
    - Documentation
  - **Engineering:**
    - All about developing products using well known and well defined principles and methods

**Software Engineering:**
Application of a systematic and disciplined approach to the development, operation and maintenance of software
Overview

**Software Evolution:**
Process of developing a software product using software engineering principles and methods.

Includes:
- Initial development
- Maintenance of code and documentation
- Updated of code and documentation
till the desired software product is developed satisfying the requirements.
Software Paradigm:
Methods and steps taken while designing the code.

Consists of:
- Software development paradigm:
- Software design paradigm
- Programming paradigm
Characteristics of good software

- A measurement of what the software does and how well can be used.
- Must satisfy:
  - Operational criteria: how well the software works
    - Usability
    - Efficiency
    - Correctness
  - Transitional criteria: important when the software is moved to another platform
    - Portability
    - Interoperability
  - Maintenance criteria: software capabilities to maintain itself
    - Modularity
    - Maintainability
SDLC
Software Development Life Cycle
Software Development Life Cycle (SDLC)
Well-defined, structured sequence of stages in software engineering to develop the software product

- Provides a serie of steps to be followed to design and develop a software in an efficient way.
- Includes:
  - 1. Communication
  - 2. Requirement gathering
  - 3. Feasibility study
  - 4. System analysis
  - 5. Software design
  - 6. Coding
  - 7. Testing
  - 8. Integration
  - 9. Implementation (installation)
  - 10. Documentation
  - 11. Maintenance
SDLC – Communication
Initial request for a desired software product. Negotiation of terms …

SDLC – Requirements Gathering
Discussions with stakeholders to bring out as much information as possible on the requirements (user, system and functional) of the software product.

SDLC – Feasibility Study
Can the software be designed to fulfill all requirements? Is the project financially, practically and technologyically feasible?
SDLC – System Analysis

Analyzes the scope of the project and plans the schedule and resources accordingly

- Includes among others:
  - understanding of software product limitations
  - learning system related problems or changes to be done in existing systems beforehand
  - identifying and addressing the impact of project on organization and personnel etc.
SDLC – Software Design

Design of the product taking into account the whole knowledge of requirements and the system analysis report

- Outputs:
  - meta-data and data dictionaries
  - logical diagrams
  - data-flow diagrams
  - prototype
  - in some cases pseudo codes
SDLC – Coding (programming phase)
Consists of writing program code in the suitable programming language and developing error-free executable programs efficiently.

SDLC – Testing
Evaluation of the software against the requirements gathered from users and system specifications.

- Different levels of testing:
  - Test while coding: modules, interfaces, program
  - Product testing: integration of components
  - In-house testing
Software Development Life Cycle

SDLC – Integration
The software might be integrated with external libraries, databases or external software.

SDLC – Implementation (installation)
Installation of the software application on user machines. Includes post installation and final configurations (if needed).

SDLC – Documentation
User guide, installation guide, code comments…

SDLC – Maintenance
The software product has to be reviewed and maintained periodically by updating the code according to changes, the documentation… May face bugs.
Project management
Software Project Management
Science of planning and leading software projects

- Includes:
  - Managing people activities
    - Managing human resources
    - Meetings with stakeholders
    - Maintain communication among team members
    - ...
  - Managing project activities
    - Define and set up the project scope
    - Estimate the project duration and scheduling
    - Manage the non-human project resources
    - Monitor progress and performance
    - Risk analysis
    - ...

Basics of Software Engineering
Project management

- Project estimation
  - Might involve the following measures:
    - Software size estimation
      - KLOC (Kilo Lines of Code)
      - Function points: related to the number of user and software requirements
    - Efforts estimation: in terms of
      - Personnel requirements
      - Man-hour required to produce the software
    - Time estimation
    - Costs estimation: should consider
      - Size of the software
      - Software (tools, licenses, …) and Hardware requirements
      - Communication costs
      - Travel costs
      - Training and support costs
Project management

- Gantt chart
  - Manage resources
  - Estimate costs, efforts and time
  - Analysis of critical paths
- PERT (Program Evaluation and Review Technique) chart
  - Despicts project as a network diagram.

Critical Path: 83 Days
Project management

- Risks management
  - Should include:
    - Experienced staff leaving the project and new staff coming in
    - Partners leaving the project
    - Change in organizational management
    - Requirement changes or misinterpreting requirement
    - Under-estimation of required time or/and resources
  - Management process
    - Identification: brainstorming about risks which may occur
    - Categorization: into high, medium or low risk intensity and impact
    - Manage: make plans to minimize the risk and avoid/face side effects
    - Monitor: monitor early symptoms and effective steps to mitigate them.
Requirements analysis
Requirement Engineering Process
Gathering of requirements from the client: description of features and functionalities desired of the requested software product.

- Steps:
  - Feasibility study
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation
Requirements analysis

- Feasibility study:
  - The user comes with a rough idea about the functionality of the software and all features expected.
  - The analyst decides whether the proposed system is worthwhile
  - Output: feasibility report:
    - If the system can be engineered using current technology and within budget
    - If the system can be integrated with other systems that are used

- Elicitation process (requirements discovery):
  - Find out about the application domain, the services that the system should provide and the system’s operational constraints.

- Requirements specifications:
  - Will define, how the software will interact with the hardware, external interfaces, portability across platforms, maintainability, speed of recovery after crash…

- Requirements validation:
  - Checks for any ambiguous requirements
  - Can be practically implemented
  - …
Design
Design

Process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

- Output:
  - The output of this phase can directly be used in the programming and implementation phase.
  - 3 different levels:
    - Architectural Design: highest abstract version of the system. It identifies the software as a system with many components interacting with each other.
    - High-level Design: how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
    - Detailed Design: defines logical structure of each module and their interfaces to communicate with other modules.
Data Flow Diagram (DFD)
- Graphical representation of flow of data in an information system
- Depicts incoming data flow, outgoing data flow and stored data
- No information about how data flows through the system

- Components
  - Entity: Source and destination of information data
  - Process: Activities and actions taken on the data
  - Data Store
  - Data Flow: Movement (shown from the base of arrow as its source towards head of the arrow as destination)
Design

- DFD – 0 (System DFD):
  - Highest abstraction level which depicts the entire information system
- DFD – 1:
  - Basic modules in the system and flow of data among various modules.
  - Mentions basic processes and sources of information.
- DFD – 2 …:
  - Data flows inside the modules mentioned in DFD 1
  - Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.
Coding and Documentation
Coding and documentation

- The code should:
  - Be reliable
  - Be maintainable
  - Be readable
  - Not neglect performance

```
main(_){_^448&&main(~_);putchar(--_%64?32|~7[^TIME__-
_/8%8]"txiZ^(~z?-48]>>";;;==~$::199"^{2&8/_64/(&?1:8)%8&1:10};}
```
Coding and documentation

- Output:

```
!! !!!!!!!
!! !!
!! !!
!! !!!!
!! !!
!! !!
!! !!!!
```

```
17.43 R
```
Coding and documentation

- Prefer the pass-by-reference to pass-by-value.
- Avoid global variables
- Modularize the code:
  - Write your code in routines
  - Write routines in manageable blocks (paragraphs)
  - Thumb rule: you should be able to see a single function on your screen
- Use reliable libraries:
  - High optimized. Sometimes even architecture dependent optimization
  - Save development and debugging time
- Remove code redundancies
- Keep with coding guidelines:
  - Coding styles varies with languages, operation system and developers team
  - It should be defined and agreed before the programming phase
- Free/close resources
- Comment your code
Coding and documentation

● Coding guidelines:
  1. Typical naming conventions

  **C:**
  Constants: MAX_STRING, NUM_FILES, ELEMENTS…
  Trivial variables (counters) / arguments: i, n, p, …
  Local variables: iInteger, dDouble, sString, pPointer, iLongName…
  Global variables: gilInteger, gdDouble…
  Structures: tType
  Functions: rotate, draw, readOutputFile, initFunction, …
  Files: file.c, include_file.h, …

  **C++:**
  Namespaces: std, longnamespace, …
  Class: Point, ThisIsMyClass, …
  Members: mIInteger, mDDouble, mSString, …
  Member functions: like C-Functions
2. Indentation: tabs or spaces? Brackets in new lines or at the end of line?

The whole idea behind indentation is to clearly define where a block of control starts and ends. Spacing of 8, 4 or 2 characters is meaningless. Indent:

- Functions, methods, routines…
- loops, conditionals…

```plaintext
if (x == y) {
    // do this
} else if (x > y) {
    // do that
} else {
    // do neither this nor that
}
```
3. **Whitespaces:** between parenthesis, before/after parenthesis, assignments?

```c
int f (int a)
int f(int a)
int f ( int a )
```

```c
int i=0;
int *p=NULL;
int i = 0;
int* p = NULL;
int i=0, *p=NULL;
...
```

4. **Control and loops statements**

```c
for (int i=0; i<N; i++)
for (i=0; i<N; i++)
for(i = 0; i < N; i++)
```

```c
if (p || a)
if (p!=NULL || a!=0)
if ((p!=NULL) || (a!=0))
if (p != NULL || a != 0)
```
5. Comments: blocks or inline

```c
/* This is a very long comment in which I
will explain what is going on in this function
and also what should it receive and return */

// This is a very long comment in which I
// will explain what is going on in this function
// and also what should it receive and return

p = doPostprocessing(...); //This function should create the image

//This function should create the image
p = doPostprocessing(...);
```
Coding and documentation

6. Including / using headers

```c
#include <errno.h>
#include "functions.h"
#include "headers.h"
#include <stdio.h>
#include <stdlib.h>
```

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
Coding and documentation

- Well documented project should involve:
  - Requirements documentation:
    - Functional and behavioral description of the software
    - Used by the software designer, developer and test teams
  - Software design documentation:
    - Necessary information needed to build the software.
    - It contains the software architecture, software design details, dataflow diagrams and databases design if needed.
  - Technical documentation:
    - Created by the developers team while writing the code
    - Represent information about the code
    - Increase understanding between programmers working on the same code
  - User documentation
    - Explains the users the software product and how should it work.
    - Might include software installation procedure, how to run the software, how to use, uninstallation method, licenses...
Testing
Testing

- All software crashes: statistics says, that every third line of C-code generates an error.

Testing
Evaluation of the software against the requirements gathered from users and system specifications.

- Consists of
  - Software validation: satisfies the software the user requirements?
  - Software verification: meets the software the proper specifications and methodologies?

- The results of the testing phase should be reported to the application manager.
Testing

● **Software validation**
  – Does the developed software fulfill the user needs?
  – The software will be validated if it satisfies the user requirements for which it was made
  – Carried out at the end of the SDLC

● **Software verification**
  – Are we developing following the design specifications?
  – Process to confirm if the software meets the business requirements and if it is/was developed using the proper specifications and methodologies
  – Ensures that the product meets the design specifications
Testing

- Do not use your main application as a testing ground! Instead, write little helper applications in order to familiarize yourself with new features.

- Two flavors of testing
  - Manual testing
    - Perform without the help of an automated testing tool
    - The software tester develop test cases for testing different several program functionalities which should either crash or successful finish the execution
    - Time and resource consuming
  - Automated testing
    - Testing procedure perform with aid of an automated testing tool.
    - Example: testing if a certain webserver can react to a load of millions of connections
Testing

● Approaches
  – Black-box: without taking into account the implementation. A set of input values are given and respectively the desired results.
  
  – While-box: tests functionality and its implementation to improve code efficiency or structure:
    – Control-flow tests: tests cases which cover all statements and branch conditions (being true and false).
    – Data-flow tests: covers all the data variable included in the program. It tests if the variables were all declared and defined, where they where changed and used.
Testing

● Test levels
  – Unit tests
    – Tests on the units (functions, modules…) to know if they are error free.
    – White-box tests
  – Integration tests
    – Find out if the units once integrated together still work error free.
    – Argument passing and data updating are tested.
  – System tests (once the software is compiled as a product)
    – Tests whether it works correctly as a whole.
    – Functionality, performance and security should be tested
  – Acceptance tests (before the software arrives the user)
    – GUI and documentation are tested.
    – The software may work but the end user might not like the appearance.
    – Alpha (team internal tests) and Beta tests (user tests)
  – Regression tests
    – Tests for detect negative impacts of new added code, feature or functionality
Maintenance
Maintenance

SDLC phase which stands for all the modifications and updates done after the delivery of the software product

- Types of maintenance:
  - Corrective: includes modifications and updates done in order to correct or fix problems, which are either discovered by user or concluded by user error reports
  - Adaptive: includes modifications and updates applied to keep the software product up-to date
  - Perfective: includes modifications and updates done in order to keep the software usable over long period of time. Includes new features, new user requirements for refining the software and improve its reliability and performance
  - Preventive: includes modifications and updates to prevent future problems of the software which can cause serious issues in future
Maintenance

Every new modification or update must be:

- Identified: by user requirement, logs...
- Analyzed: impact on the system including safety and security implications
- Designed: against requirement specifications
- Implemented:
- Tested: tests between new modules and the system and as a whole (regression tests)
- Delivered: changes deployed over the organization (updates package)