software engineering

Tuğberk Kocatekin
Istanbul Arel University

IDE (Integrated Development Environment)

- These systems have multiple tools for developing software
  - Editors
  - Compilers
  - Debugging tools
  - ..
- Some examples are
  - PyCharm
  - PHPStorm
  - Visual Studio
  - Android Studio
  - Eclipse

Software Life Cycle

- Once software is developed, it enters a new cycle where it is *used* and *maintained*.
- This is similar to other products, however the *maintenance* in software is **updating**.
- It must be well-designed and well-documented. This is very important for maintenance. Otherwise, while there is good intention, the outcome can be bad.

Traditional Development Phases

- Requirements analysis
  - Takes input from the **stakeholders** of the proposed system.
  - Identifying the problem to be solved.
- Design
  - Creating a plan for the construction of the proposed system.
  - The internal structure of the software is established here.
  - A lot of *diagrams* and *models* are used.
- Implementation
  - Actual writing of the programs: creating data files, creating databases.
  - In the narrowest sense: A programmer here implements the design done by **software analysts**.
- Testing
  - Traditionally it is done at the end, however in new software development cycles, testing should be done at every step to ensure quality.

Software Engineering Methodologies

- Waterfall model
  - Entire requirements specifications are done before beginning design. So, every step must be done before going into the next step.
- Incremental model
  - Software system is constructed in increments. First, you make a very simple version with limited functionality. It is tested and later new features are added and tested again.
- Iterative model
  - Similar to incremental. Here, version are refined.
- Agile methods
  - More to come

Incremental and Iterative

- Similar but not the same.
  - Incremental model tries to *extend* every version by adding new features and make into a larger version.
  - Iterative model encompasses the concept of *refining* each version.
- RUP: Rational Unified Process
  - Software development paradigm which redefines the steps in development phase and provides guidelines. Developed and marketed by IBM.
  - A non commercial version is called **unified process**.
- Prototyping
  - Incomplete versions of software.
  - **Evolutionary prototyping** for incremental models. It evolves into complete and final system.
  - **Throwaway prototyping** for iterative models. Old versions are discarded in favor of fresh versions.

Open Source Development

- An author writes the initial version of the software, puts the source code and its documentation.
- Others can download and change, edit the code. They can modify or improve the software for their own purposes.
- Report these changes to the author.
- Author checks and decides whether to accept that change.
- …

Agile methods

- Early and quick development.
- Reduced emphasis on requirements analysis and design.
- Software is developed incrementally by
  - repeated daily cycles of informal requirements analysis, design, implement and testing
  - new expanded versions are regular
- Consists of **sprints**.
  - A set period of time which specific work needs to be completed and made ready for review.
  - Takes between 15-30 days.

Agile development

- In a product development there are a lot of elements. Not just developers. Developers are part of a **product team**.
  - Product owner/manager
  - Business Analyst
  - Designer, UX designer
  - Testers & Developers

Kanban (visual signal)

- Popular framework to implement agile.
- Kanban board
  - Backlog
  - To-Do
  - Doing
  - Done
- Every task can be seen by everyone and can be tackled by everyone.
- Trello is a free(& paid) web application for easy Kanban boards.

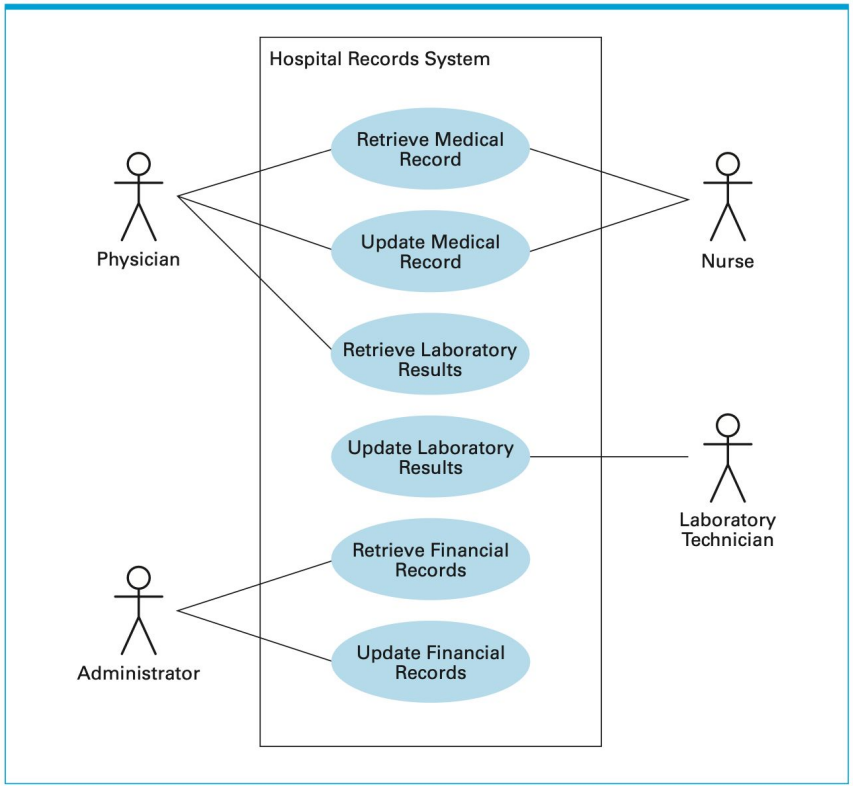
UML: Unified Modeling Language

- A general purpose modeling language which aims to provide a standard way to visualize the design of a system.
- Useful for communicating with non-programmers.

Example diagram

- Stick figures are called **actors**.
- Oval figures are called **use cases**.
- This use case diagram depicts the *Hospital Records System*.

Figure 7.9 A simple use case diagram



Design Patterns

- Reusable code solution to a commonly occurring problem. (OOP)
- Define common solutions to problems which the team can use together.
- There are a lot of design patterns to be selected based on the problem you are trying to solve.
- Some examples:
  - Factory method, iterator, adapter, bridge, builder, singleton, etc.

Testing

- Most software is verified by **testing**.
  - However, this may not be true at all cases. Testing every possible case is almost impossible.
- However, it is observed that small number of modules within a software system tend to be more problematic. Therefore, identifying these modules and testing them more thoroughly is better than testing all models in a uniform, less-thorough manner.
  - Pareto principle
  - Probably 20% of the software will create 80% of problems.

Unit testing

- Unit is the smallest testable part of a software.
- Unit tests are usually *automated tests* written by software developers to ensure that a unit works as intended.
- It isolates and show that individual parts are working correctly.
- When you code a task, you use the unit tests before pushing that code.
- It fix bugs early and therefore save costs.

Testing methodologies

- Basis path testing
  - Develop a set of test data that insures that every instruction in the software is executed at least once. It may be impossible to test every other path, but it can make sure that every instruction works at least once.
- Glass-box testing
  - Tests internal structure or working of the software as opposed to its functionality. Internal perspective of the system is used to create test cases.
- Black-box testing
  - Performed from the user’s point of view. Don’t care about the task, checks whether the software is performing correctly in terms of accuracy and timeliness.

Beta & Alpha Testing

- Part of *black-box testing*.
- A preliminary version of the software is given to an audience with the goal of learning how it will perform in real-life situations.
- Done before final version is released.
- When done within an organization: **alpha testing**
- When done in user's environment: **beta testing**

Documentation

- Very important in software development. Developing is a part, but **maintaining** is also important.
- Serves three purposes:
  - User documentation
  - System documentation
  - Technical documentation

User documentation

- Explain the features of the software and describe how to use them.
- Written to the **user** of the software.
- Many software companies hire technical writers to write these because it is very important for sales etc.
- Sometimes they even provide a preliminary software to writers so that they can write a book/documentation when the product is ready.

System documentation

- Describes the internal of the software.
- Very important for maintenance.
- Some companies have specific:
  - Naming conventions
  - Indentation conventions (PEP)
  - Documentation conventions

Technical documentation

- Describe how the software should be installed and serviced.
  - Adjusting operating parameters, installing updates, reporting problems back to developers.
- Analogous to documentation provided to mechanics in the automotive industry. It does not include how the software is designed or implemented, or how to use/operate it. It describes how to service it.

Software license

- A legal agreement between the owner and user of a software product that grants the user certain permissions to use the product.
- Several licenses in the open-source
  - Permissive License
    - Grants use rights and proprietization
    - MIT, Apache
  - Copyleft
    - Grants use rights but not proprietization
    - GPL, AGPL