Project Management

Revision control: process of managing multiple versions of a piece of information.

RCS: Revision control software, store the history of the working directory as a series of **commits**.

- can be done in centralized way or distributed way

Centralized RCS (CRCS)

- uses a central remote repo that is shared by the team
- interact directly with this central repository
- CVS and SVN support **only** this model

Distributed RCS (DRCS)

- allows multiple remote/local repos **working together**
- workflow can vary from team to team
- Git and Mercurial support this

Pros of use an **automated** revision control tool for a project

- track the history and evolution of your project
- makes it easier for you to collaborate
- helps you to recover from mistakes
- helps you to work simultaneously on, and manage the drift between, multiple versions of your project.

Repository: database that stores the revision history tracked by an RCS software

Working directory: root directory revision-controlled by Git.

Committing saves a snapshot of the current state of the tracked files in the revision control history

Each **commit** in a repo

- is a **recorded point** in the history of the project
- is **uniquely** identified by an **auto-generated hash**
- can **tag** a specific commit with a more easily identifiable name
- **diff** to see what changed between two points of the history
- To restore the state of the working directory at a point in the past, you can **checkout** the commit in concern.

Remote Repositories: repos that are hosted on remote computers

- **pull (or fetch)** from one repo to another, to receive new commits in the second repo
- A repo can work with any number of other repositories as long as they have a shared history
- A **fork** is a remote copy of a remote repo.

Branching is the process of evolving multiple versions of the software in **parallel**.

Merge conflicts happen when you try to merge two branches that had changed the **same part** of the code and the RCS **cannot decide which changes to keep**.

Forking workflow

- the 'official' version of the software is kept in a remote repo designated as the 'main repo'.
- All team members fork the main repo and create pull requests from their fork to the main repo.

WBS: **Work Breakdown Structure** depicts info. about **tasks** and their **details** in terms of **subtasks**.

- effort is measured in **man hour/day/month**
- All tasks should be **well-defined** [clear as to when the task will be considered done.]

Milestone: **end of a stage** which indicates significant progress.

- Each intermediate product release is a milestone.
- If not practical to have a very detailed plan→use a high-level plan for the whole project and a detailed plan for the **next few milestones**.

Buffer: time set aside to absorb any unforeseen delays

Do not inflate task estimates to create hidden buffers; have explicit buffers instead.

Reason: With explicit buffers, it is easier to detect incorrect effort estimates which can serve as feedback to improve future effort estimates.

Issue trackers (bug trackers): used to track task assignment and progress. [GitHub, SourceForge, and BitBucket]

Team Structures

SDLC process models

Software Development Lifecycle (SDLC): different stages of software development. such as requirements, analysis, design, implementation and testing.

Software Development Lifecycle Models (software process models): approaches that describe different ways to go through the SDLC.

Sequential Model [waterfall model]

- views software development as a **linear process**
- When one stage of the process is completed, it **produces some artifacts to be used** in the next stage.
- A strict sequential model project **moves only** in the **forward** direction
- can work well for a project that produces software to solve a **well-understood problem**
- **real-world** projects **often** tackle problems that are not well-understood at the beginning [**unsuitable** for this model]

Iterative Model

- advocates producing the software by going through several iterations.
- produces a new version of the product **each iteration**
- can be done in **breadth-first** or **depth-first** approach or **mixture**.

Breadth-first approach

- an iteration **evolves all major** components and all functionality areas in parallel
- **most** features and **most** components will be updated in each iteration
- producing a **working product** at the end of each iteration.

Depth-first approach

- an iteration focuses on **fleshing out only some** components or some functionality area.
- early iterations might not produce a working product.

<u>Agile processes</u>

eXtreme Programming (XP)

- stresses **customer satisfaction**
- aims to **empower developers** to confidently respond to changing customer requirements
  [even late in the lifecycle]
- emphasizes **teamwork**
- aims to improve a software project in five essential ways: communication, simplicity, feedback, respect, and courage
- has a set of **simple rules**
- Pair programming, CRC cards, project velocity, and standup meetings are some interesting topics related to XP

Scrum: a process **skeleton** that contains sets of practices and predefined roles. The main roles in Scrum are:

- **Scrum Master:** maintains the processes
- **Product Owner**: represents the stakeholders and the business
- **The Team**, a cross-functional group who do the actual analysis, design, implementation, testing, etc.

A Scrum project is divided into iterations called **Sprints**

- basic unit of development
- tend to last between one week and one month
- are a timeboxed (i.e. restricted to a specific duration) effort of a constant length.
- preceded by a planning meeting
- team creates potentially deliverable product increment during each sprint

Key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need

Daily Scrum

- is another key scrum practice
- each team member answers the following three questions during the daily scrum
  - What did you do yesterday?
  - What will you do today?
  - Are there any impediments in your way?
- meeting [Morning] is not used as a problem-solving or issue resolution meeting.
- Issues raised are taken offline and usually dealt with by the relevant subgroup immediately after the meeting.

Git convention

- Every commit must have a well-written commit message subject line.
- Try to limit the subject line to 50 characters (**hard limit: 72 chars**)
- Use the **imperative** mood in the subject line.
  ["Add" instead of "Added" or "Adds"]
- **Capitalize** the first letter of the subject line.
- **Do not end t**he subject line **with a period.**
- You can use `scope: change` format or `category: change`
- Commit messages for **non-trivial commits** should **have a body** giving details of the commit.
- Separate subject from body with a blank line.
- **Wrap the body at 72 characters**.
- Use blank lines to separate paragraphs.
- Use bullet points as necessary.
- **Explain WHAT, WHY, not HOW.**
- If your description starts to **get too long**, that's a sign that you probably **need to split up your commit** to finer grained pieces.