CT417

Software Engineering III

Name: Andrew Hayes
E-mail: a.hayes18@universityofgalway.ie
Student ID: 21321503

2024–09–15

Contents

1 Introduction

1.1 Lecturer Contact Details

- Dr. Effirul Ramlan.
- Email: `effirul.ramlan@universityofgalway.ie`.
- Will attempt to reply to emails immediately between the hours of 09:00 & 20:00 from Week 01 to Week 12.
- Discord server: `https://discord.gg/CRAtHv9uNg`.

1.2 Grading

- Continuous Assessment: 40%.
  - You will work in pairs on a software project with three key submissions across the 12 weeks. Each deliverable will align with the topics covered in the course up to that point, allowing for continuous progress assessment.
  - AS-01: Set up musicFinder and configure the CI/CD pipeline (Week 4).
  - AS-02: Testing, Security, & Expanded Application (Week 8).
  - AS-03: Refactoring & Application Deployment.
- Final Exam: 60%.
  - Typical 2-hour exam paper covering materials from Week 1 to Week 12, with nothing out of the ordinary – "You can be sure of that".

2 Revision

2.1 What is Software?

Software consists of:

i. Instruction (computer programs) that when executed provide desired features, function, & performance.

ii. Data structures (Arrays, Objects, Lists, Dictionaries, Maps, etc.) that enable programs to manipulate information.

iii. Descriptive information in both hard copy & virtual format describing the operation & use.

2.2 Functional vs Non-Functional Requirements

Functional Requirement	**Non-Functional Requirement**
Describes the actions with which the user's work is concerned	Describes the experience of the user while doing the work
A feature or function that can be captured in use-cases	A global constraint (and therefore difficult to capture in use-cases)
A behaviour that can be analysed via sequence diagrams or state machines	A software quality
can be usually traced back to a single module / class / function	Usually cannot be implemented in a single module or even program

Table 1: Functional vs Non-Functional Requirements

Typical non-functional requirements include: availability, maintainability, performance, privacy, reliablility, scalability, & security.

2.3 What is Software Engineering?

Software Engineering is the field of computer science that deals with the building of software systems that are so large or so complex that they are built by a team or teams of engineers. Software Engineering encompasses a process, a collection of methods, & an array of tools that allow professionals to build high-quality software.

DevOps outlines a software development process that increases the delivery of higher quality software by integrating the efforts of the development & IT operation teams.

$$\text{DevOps} = \text{Software Engineering} + \text{IT Operations}$$

2.4 What are Software Development Life Cycles?

Software Development Life Cycles (SDLC) refers to a process used by software engineers to design, develop, & test software. Each approach focuses on a different aspect of development, from planning to continuous improvement.

2.5 What is a Framework?

A **software framework** is an abstraction in which common code providing generic functionality can be selectively overridden or specialised by user providing specific functionality.

Low-code is a method of designing & developing applications using an intuitive GUI & embedded functionality that reduce traditional professional code writing requirements. **No-code** is similar to low-code, but for non-technical business users as it allows them to develop software / applications without having to write a single line of code.

2.6 Agile & DevOps

2.6.1 What is Agile?

Agile is a method of software development consisting of:

- **Iterative & Incremental Development:** Software is developed in small, workable increments.
- **Customer-Centric:** Constant feedback from customers to refine requirements.
- **Frequent Delivery:** Rapid releases of smaller, functional product versions.
- **Adaptability:** Agile responds to change quickly

2.6.2 Agile Principles

- **Individuals & Interactions:** over processes & tools.
- **Working Software:** over comprehensive documentation.
- **Customer Collaboration:** over contract negotiation.
- **Responding to Change:** over following a plan.
- **Quote:** "The highest priority is to satisfy the customer through early & continuous delivery of valuable software."

2.6.3 Agile Frameworks

Agile methodologies & frameworks include:

- **Scrum:** Divides work into sprints (2-4 weeks) with regular stand-ups & reviews.
- **Kanban:** Focuses on visualising workflow & limiting Work-In-Progress (WIP).
- **XP (eXtreme Programming):** Emphasises technical excellence & frequent releases.
- **Lean Development:** Focuses on minimising waste & maximising value.

2.6.4 What is DevOps?

DevOps is a culture & set of practices that integrated development (Dev) & operations (Ops). It involves collaboration & automation between developers & IT operations for faster delivery of high-quality software. It also involves continuous integration/continuous delivery (CI/CD) to automate code testing & deployment.

$$\text{DevOps} = \text{Development} + \text{Operations}$$

2.6.5 DevOps Core Practices

DevOps core practices include:

- **CI/CD Pipelines:** Automating the building, testing, & deployment of code.
- **Infrastructure as Code (IaC):** Managing infrastructure through code (e.g., Terraform, Ansible).
- **Monitoring & Logging:** Ensures system reliability through real-time tracking & analysis.
- **Collaboration & Communication:** Cross-functional teams sharing ownership of development & operations tasks.

2.6.6 Key Differences between Agile & DevOps

Agile	**DevOps**
Focus on frequent customer feedback	Focus on collaboration between Dev & Ops teams
Iteration done through iterative cycles	Iteration done through rapid feedback loops
Scope of smaller, incremental changes	Scope of large-scale projects
Uses task management software (e.g. Jira)	Uses automation tools (e.g. Jenkins)
Scrum, XP frameworks	Kanban, DevOps lifecycle frameworks

Table 2: Key Differences between Agile & DevOps

Agile focuses on iterative development & customer feedback, with a short feedback loop. **DevOps** focuses on automating delivery, collaboration, & integration between Dev & Ops teams, integrating the entire process for faster releases.

2.6.7 Why DevOps Complements Agile

Agile improvements development velocity, but DevOps extends the concept to deployment & maintenance. Both are customer-focused, but DevOps ensures rapid & reliable deployment in addition to development. DevOps fills gaps Agile doesn't cover, like operations, infrastructure, & automation. Agile helps development teams iterate & adapt to changing requirements, while DevOps bridges the gap between developers & IT operations.

2.6.8 Benefits of Agile & DevOps

- Faster, more frequent delivery of features.
- Improved communication & collaboration between teams.
- Reduced risk of deployment errors.
- Ability to adapt to customer feedback & market changes rapidly.
- Higher-quality software & reduced time-to-market.

3 Version Control

3.1 What is Version Control?

Version Control is a system that records changes to a file or set of files over time, allowing you to recall or access specific versions at a later date. It is also known as *revision control* or *source control*. It allows you to keep track of changes, by whom, & when they occurred. Some of the popular version control programs include Git, CVS, Subversion, Team Foundation Server, & Mercurial.

It allows us to:

- Backup the source code and be able to rollback to a previous version.
- Keep a record of who did what and when (know who to praise & who to fire).
- Collaborate with the team (know who to praise & who to fire).
- Troubleshoot issues by analysing the change history to figure out what caused the problem.
- Analyse statistics such as who is being the most productive etc.

3.2 What files should be checked in to Version Control?

Any file that influences the software build should be checked into version control. This includes configuration files, file encodings, binary settings, etc. Furthermore, anything that is needed to setup the project from a clean checkout / fork should also be included in the version control, such as source code, documentation, manuals, image files, datasets, etc.

You should not check in any binary files such as JAR files or any other "build" files, any intermediate files from build / compilation such as `.pyc` or `.o` files, any files which contain an absolute path, or personal preference / personal settings files.

3.3 Centralised Version Control – Subversion

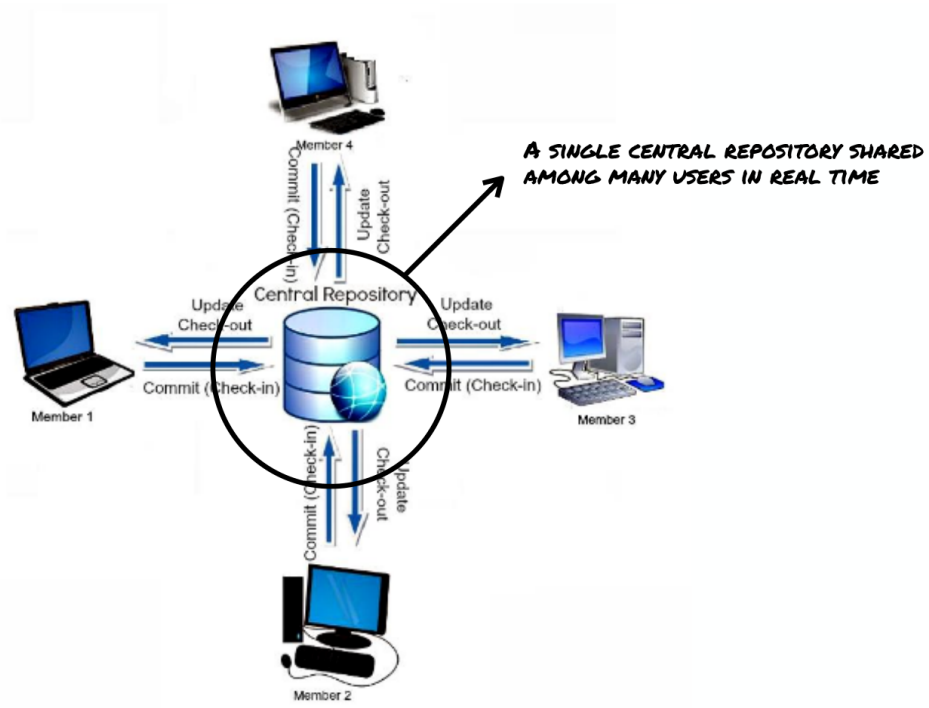

Figure 1: Centralised Version Control System Diagram

Subversion is a centralised version control system in which code is centralised in a repository which can be checked out to get a working copy on your local machine. In general, you don't have the entire repository checked out in Subversion, just a specific branch. Changes are committed back to the central repository, "normally" with useful comments, and a change log is maintained of who did what & when.

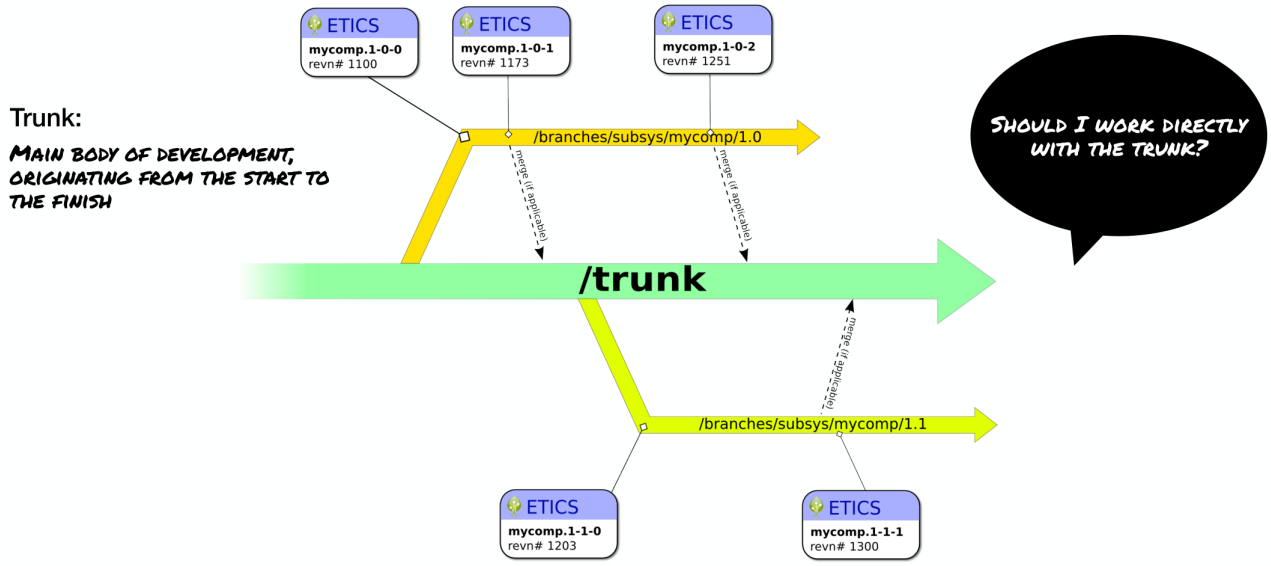

Figure 2: Trunk in Subversion

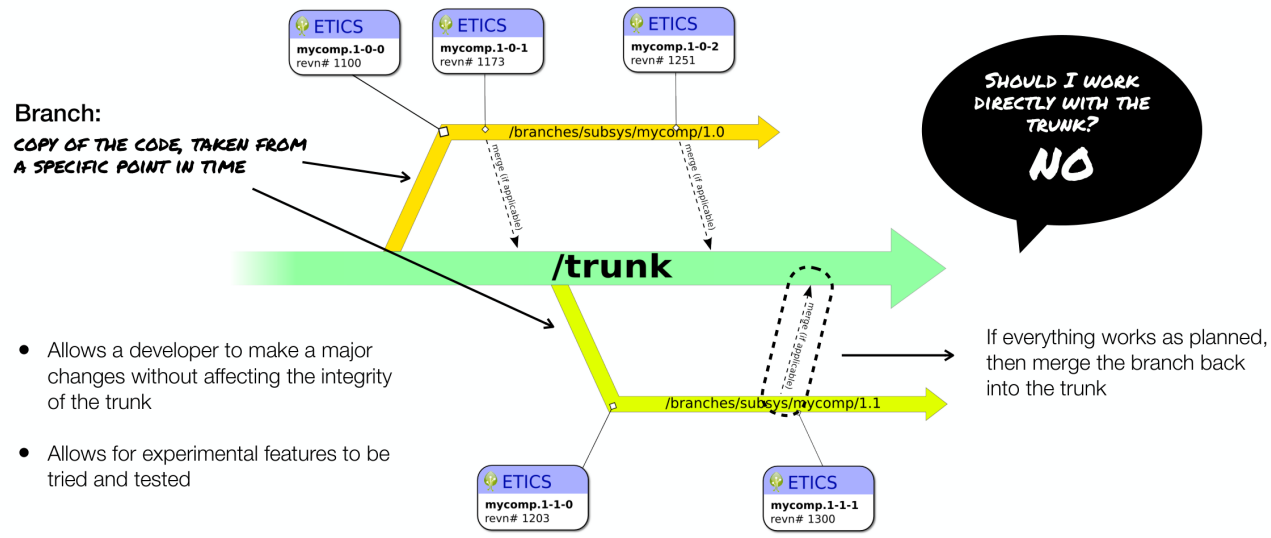

Figure 3: Branching in Subversion

When you check out the project in subversion, you will get the `HEAD` revision. When you invoke the command `svn update`, you are updating your local copy to the `HEAD` version as well. Branches should eventually be merged back into the trunk with `svn commit`. The trunk must build afterwards. The commit is a process of storing changes from your private workplace to the central server. After the commit, your changes are made available to the rest of the team; other developers can retrieve these changes by updating their working copy. Committing is an **atomic operation**: either the whole commit succeeds, or it is entirely rolled back – users never see a half-finished commit.

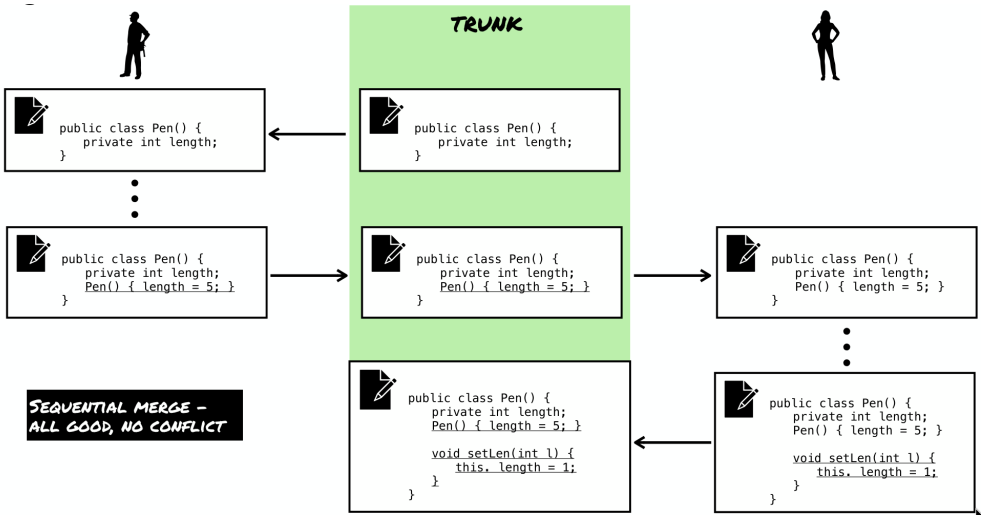

Figure 4: Sequential Merge in Subversion

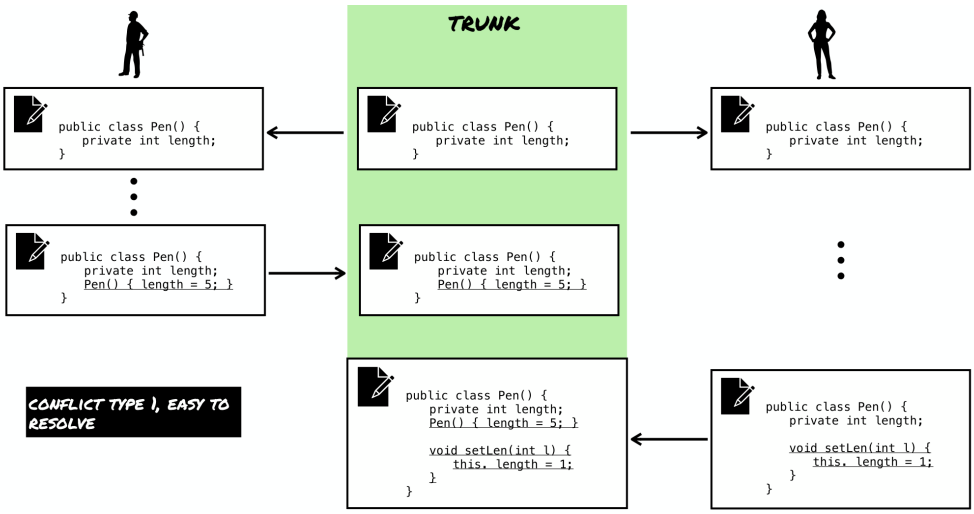

Figure 5: Type 1 Merge Conflict in Subversion

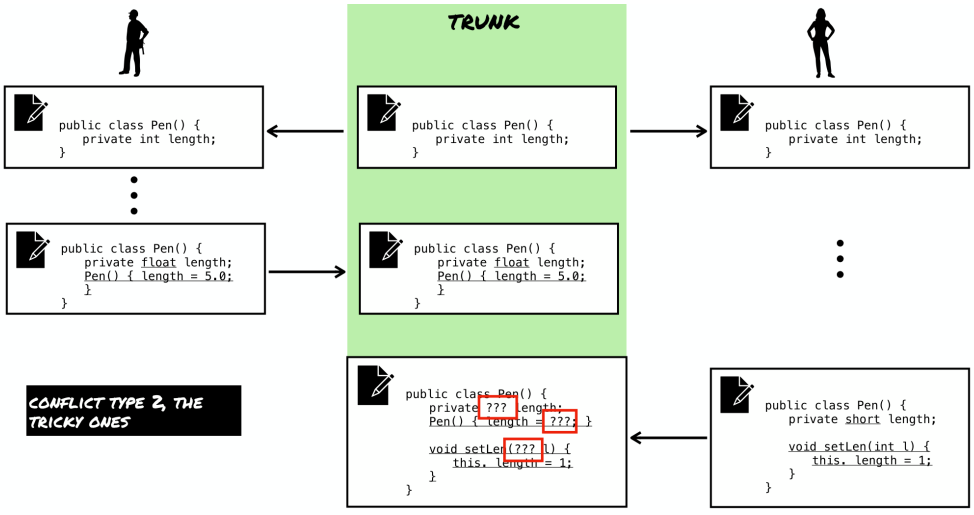

Figure 6: Type 2 Merge Conflict in Subversion

3.4 Distributed Version Control – Git

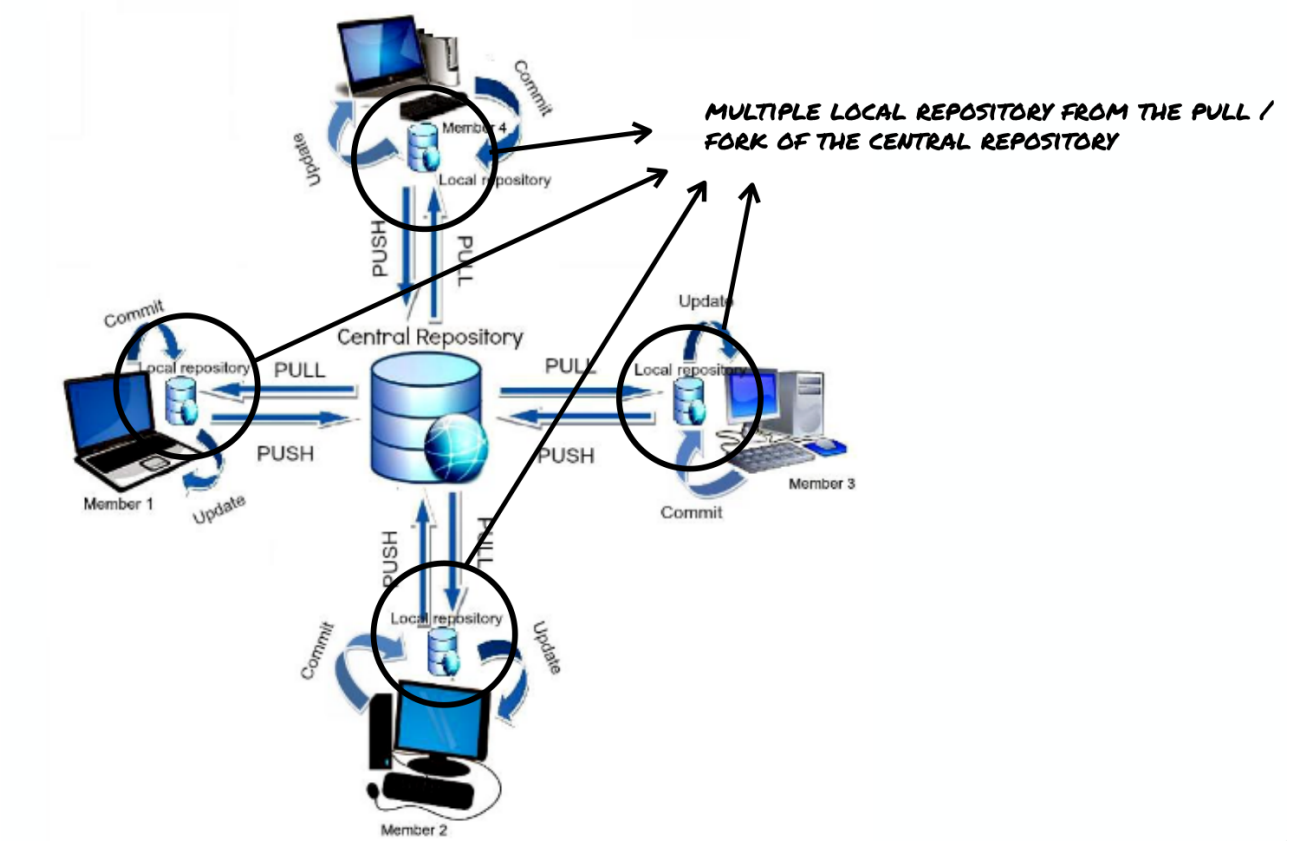

Figure 7: Distributed Version Control System Diagram

Git encourages branching for every feature, regardless of size. After successful completion of the new feature, the branch is merged into the trunk. Each developer gets their own local repository, meaning that developers don't need a network connection to commit changes, inspect previous version, or compare `diffs`. If the production trunk / branch is broken, developers can continue working uninhibited.

3.4.1 GitHub

GitHub is a web-based hosted service for Git repositories. It allows you to host remote Git repositories and has a wealth of community-based services that makes it ideal for open-source projects. It is a publishing tool, a version control system, & a collaboration platform.

3.4.2 Git Commands

- `git clone`: download ("clone") the source code from the remote repository.
- `git fetch`: fetches the latest version from the repository that you've cloned but doesn't synchronise with all commits in the repository.
- `git pull`: pulls the latest version from the repository that you've cloned and synchronises with all commits in the repository. Equivalent to running `git fetch` & `git merge`.
- `git push`: pushes the changes that you have committed to your local branch to the remote repository.

3.4.3 Pull Requests

A **pull request** is when you ask another developer to merge your feature branch into their repository. Everyone can review the code & decide whether or not it should be included in the master branch. The pull request is an invitation to discuss pulling your code into the master branch, i.e. it is a forum for discussing changes.