Faculty of Information Technology
Department: Software Engineering
GROUP ASSIGNMENT 4 (Group A)
Course name: BEST PROGRAMMING PRACTICES AND DESIGN PATTERNS

GROUP MEMBERS:
MUHOZA GYSSAGARA Prince 24417
MUTABAZI Amos 24432
HABINKA Raissa 24345
NENGO Ally 24620

QUESTION 1

a) Briefly describe Version Control Systems (VCS)

A **Version Control System (VCS)** is a tool that helps developers manage changes to source code or files over time. It allows tracking revisions, reverting to previous versions, and collaborating with others by managing concurrent updates efficiently.

b) Types of Version Control Systems

i. Local Version Control Systems

- **Architecture:** Changes are tracked locally on the user's machine using a simple database (like RCS - Revision Control System).
- **Example:** RCS (Revision Control System)
- **Description:** Stores patch sets in a special format on disk; developers manually check in/check out files.
- **Limitation:** No collaboration — not suitable for teams as there's no central or shared location.

ii. Centralized Version Control Systems (CVCS)

- **Architecture:** A single central server holds all versions; users check out/update files from this server.
- **Example:** Subversion (SVN), CVS
- **Advantages:** Easy to manage; centralized backups; better than local VCS for collaboration.
- **Limitations:** If the central server fails, access is lost; merging changes can be complex.

iii. Distributed Version Control Systems (DVCS)

- **Architecture:** Each developer clones the full repository including history. No need to be connected to a central server for most operations.
- **Examples:** Git, Mercurial
- **Advantages:** Full local history; supports offline work; more robust to failure; branching/merging is easier.
- **Limitations:** Slightly higher learning curve.

c) Benefits of Version Control Systems

- **Collaboration:** Multiple users can work on the same project simultaneously.
- **History Tracking:** Maintains history of changes for auditing and rollback.
- **Branching and Merging:** Allows isolated changes, experimentation, and controlled integration.
- **Backup:** Safe, recoverable storage of project files.
- **Accountability:** Tracks who made changes and when.
- **Conflict Management:** Helps detect and resolve conflicting changes.

QUESTION 2

a) How to Install and Configure SVN

On Windows:

1. Download and install **TortoiseSVN** from https://tortoisesvn.net/
2. Restart your PC.
3. Right-click in any folder to see SVN options (e.g., checkout, commit).
4. To create a repository:
   - Create a new folder.
   - Right-click → "TortoiseSVN" → "Create Repository Here".
   - Choose a folder structure (default recommended).
   - Done.

On Linux:

```
sudo apt update
sudo apt install subversion
```

To create a repository:

```
svnadmin create /path/to/repo
```

b) Adding SVN to IDE (e.g., Eclipse or NetBeans)

In Eclipse:

1. Go to **Help > Eclipse Marketplace**.
2. Search for **Subclipse** or **Subversive** plugin and install.

3. Restart Eclipse.
4. Open **SVN Repository Exploring** perspective.
5. Add your repository URL.
6. Check out project into your workspace.

In NetBeans:

1. SVN support is often pre-installed.
2. If not: Go to **Tools > Plugins**, search for **Subversion**, and install.
3. Restart NetBeans.
4. Use **Team > Subversion > Checkout** to connect to your repo.

c) Key VCS Terminologies

Term	Explanation
Repository	A central location where all versions and history of a project are stored
Working Copy	Local version of files that a user works on, usually checked out from a repo.
Revision	A specific version/snapshot of the project after a commit.
Commit	Saving changes to the repository, creating a new revision.
Update	Bringing your working copy up to date with changes from the repository.
Checkout	Copying files from the repository to your local machine.
Branch	A diverging path of development used for features or experiments.
Tag	A snapshot of the repository at a particular point, often used for releases.

Merge	Integrating changes from one branch into another.
Conflict	Happens when different changes affect the same lines of a file in different versions.
Diff	A comparison of differences between two versions of a file.
Log	History of commits made to a repository or file.
Lock	Temporarily prevents others from editing a file to avoid conflicts.
Unlock	Releasing the lock on a file so others can edit it.

Explanations with examples on git version control, and how to run them in the terminal

Term	Git Usage Example / Explanation
Repository	A Git project folder tracked with version control. *Created using:* `git init` or `git clone <url>`
Working Copy	The actual files in your project directory. You modify these before staging and committing.
Revision	Each commit in Git is a unique revision, identified by a SHA-1 hash. *Example:* `e83c5163316f89bfbde7d9ab23ca2e25604af290`

Commit	Saves changes to the local repo. *Command:* `git commit -m "Added login feature"`
Update	In Git, it's like `git pull` — updates local repo with remote changes.
Checkout	Switch to another branch or restore files. *Command:* `git checkout develop` or `git checkout HEAD~1 file.txt`
Branch	A pointer to a snapshot of your changes. *Create with:* `git branch new-feature`
Tag	Marks a specific commit, usually for releases. *Command:* `git tag v1.0`
Merge	Combines changes from one branch into another. *Command:* `git merge feature-branch`
Conflict	Happens when two branches modify the same line. *Resolved manually or using:* `git mergetool`
Diff	Shows what changed between commits, branches, or files. *Command:* `git diff HEAD~1`
Log	Displays commit history. *Command:* `git log`
Lock	Git doesn't use file locking like SVN, but you can use `git-lfs` to simulate it.
Unlock	Releasing a lock (again, mainly relevant in Git LFS).