INTRODUCTION TO SOFTWARE ENGINEERING

Lesson 05 – Implementation and Coding Practices

OUTLINE

1. Coding standards and best practices
2. Version control systems (e.g., Git)
3. Code reviews and pair programming

1. CODING STANDARD AND BEST PRACTICES

Coding standards are collections of coding rules, guidelines, and best practices.

The coding standards helps writing cleaner codes.

Coding rules and guidelines ensure that software is:

- **Safe:** It can be used without causing harm.
- **Secure:** It can't be hacked.
- **Reliable:** It functions as it should, every time.
- **Testable:** It can be tested at the code level.
- **Maintainable:** It can be maintained, even as your codebase grows.
- **Portable:** It works the same in every environment.

Check the webinar on how to apply a coding standard.

1. CODING STANDARD AND BEST PRACTICES

There are four key benefits of using coding standards:

- Compliance with industry standards (e.g., ISO).
- Consistent code quality (no matter who writes the code).
- Software security from the start.
- Reduced development costs and accelerated time to market.

Here are some coding standard rules:

1. Limited use of global (variables)
2. Standard headers for different modules
3. Naming conventions for local variables, global variables, constants and functions
4. Indentation
5. Error return values and exception handling conventions

1.1. LIMITED USE OF GLOBAL (VARIABLES)

Avoid using global static variables:

```
public static ArrayList<Double> prices = new ArrayList<>();
public static long numberOfStudents = 20;
```

Global variable persists in memory from loading class until end of program. The more global variables, the slower the system.

We should use global only when:

- Having constants (*public static final*) which are initialized in static block
  `final double PRICE = 9.9;`
- Having the variables *private static final* initialized in static block and exposed via getters
  ```
  private static final double PRICE = 9.0;
  public static final double getPrice() { return PRICE; }
  ```
- Creating a singleton and having the variables private final exposed via getters

1.1. LIMITED USE OF GLOBAL (VARIABLES)

- Creating a singleton and having the variables private final exposed via getters

```java
public class Singleton {
    private static final Singleton instance = new Singleton();
    private Singleton() {}
    public static Singleton getInstance() {
        return instance;
    }
}
```

1.2. STANDARD HEADERS FOR DIFFERENT MODULES

The header format must contain:

- Name of the module
- Date of module creation
- Author of the module
- Modification history
- Synopsis of the module about what the module does
- Different functions supported in the module along with their input output parameters
- Global variables accessed or modified by the module

We will take Singleton class as an example:

1.2. STANDARD HEADERS FOR DIFFERENT MODULES

```java
/**
 * Name: Singleton
 * Date: 08 January 2025
 * Author: Sreng
 * Modified: 08 January 2025
 * Description: A class that demonstrates the Singleton design pattern.
 * Functions:
 *      - getInstance(): Singleton
 * Variables:
 *     - instance: Singleton
 * */
public class Singleton {
    private static final Singleton instance = new Singleton();
    private Singleton() {}
    public static Singleton getInstance() {
        return instance;
    }
}
```

1.3. NAMING CONVENTIONS FOR LOCAL VARIABLES, GLOBAL VARIABLES, CONSTANTS AND FUNCTIONS

Naming conventions vary to each programming language.

In Java:

- except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.
- Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed. It should be short yet meaningful. The choice of a variable name should be mnemonic that is, designed to indicate to the casual observer the intent of its use.
- One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

1.3. NAMING CONVENTIONS FOR LOCAL VARIABLES, GLOBAL VARIABLES, CONSTANTS AND FUNCTIONS

In C++:

- Use upper case letters as word separators, lower case for the rest of the word in the class name.
- The first character in the class name must be in upper case.
- No underscores ('_') are permitted in the class name.
- The private attribute name in class should be prepended with the character 'm'.
- After prepending 'm', the same rules will be followed for the name as that for the class name.
- Character 'm' also precedes other name modifiers also. For example, 'p' for pointers.
- Each method/ function name should begin with a verb.
- The first character of function/ method argument names should be lowercase.
- The variable name should begin with an alphabet.
- Digits may be used in the variable name but only after the alphabet.
- No special symbols can be used in variable names except for the underscore('_').
- Pointer variables should be prepended with 'p' and place asterisk '*' close to the variable name.
- Reference variables should be prepended with 'r'.
- Static variables should be prepended with 's'.
- The global constants should be all capital letters separated with '_'.

1.4. INDENTATION

Ref: https://www.oracle.com/java/technologies/javase/codeconventions-indentation.html

Four spaces should be used as the unit of indentation.

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

1.5. ERROR RETURN VALUES AND EXCEPTION HANDLING CONVENTIONS

Error codes are numerical values that indicate the type and cause of an error that occurred during the execution of a program or a command. They're useful for debugging and troubleshooting purposes, as well as for scripting and automation.

Microsoft error return values:
https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--0-499-

Linux error codes:
https://github.com/torvalds/linux/blob/master/include/uapi/asm-generic/errno-base.h

1.5. ERROR RETURN VALUES AND EXCEPTION HANDLING CONVENTIONS

An exception is an error event that can happen during the execution of a program and disrupts its normal flow.

Java provides a robust and object-oriented way to handle exception scenarios known as Java Exception Handling.

Here some common exceptions:

- ParseException:

```
new SimpleDateFormat("MM/dd/yyyy").parse("invalid-date");
```

- NullPointerException.

```
String strObj = null;
strObj.equals("Hello World");
```

- ArrayIndexOutOfBoundsException.

```
int[] nums = new int[] {1, 2, 3};
int numFromNegativeIndex = nums[-1];
```

- StringIndexOutOfBoundsException.

```
String str = "Hello World";
char charAtNegativeIndex = str.charAt(-1);
```

1.5. ERROR RETURN VALUES AND EXCEPTION HANDLING CONVENTIONS

- NumberFormatException.

```
String str = "100ABCD";
int x = Integer.parseInt(str);
```

- ArithmeticException.

```
int illegalOperation = 30/0;
```

- IllegalArgumentException.

```
Thread.sleep(-10000);
```

2. VERSION CONTROL SYSTEMS (E.G., GIT)

(AKA revision control, source control, source code management) is the software engineering practice of controlling, organizing, and tracking different versions in history of computer files; primarily source code text files, but generally any type of file. Terminologies:

- **Branch:** (A set of files under version control may be branched or forked at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other)
- **Change:** (represents a specific modification to a document under version control)
- **Change list:** (a change list (or CL), change set, update, or patch identifies the set of changes made in a single commit)
- **Checkout:** (create a local working copy from the repository)
- **Clone:** (create a repository containing the revisions from another repository)
- **Commit:** (to write or merge the changes made in the working copy back to the repository)

2. VERSION CONTROL SYSTEMS (E.G., GIT)

Terminologies:

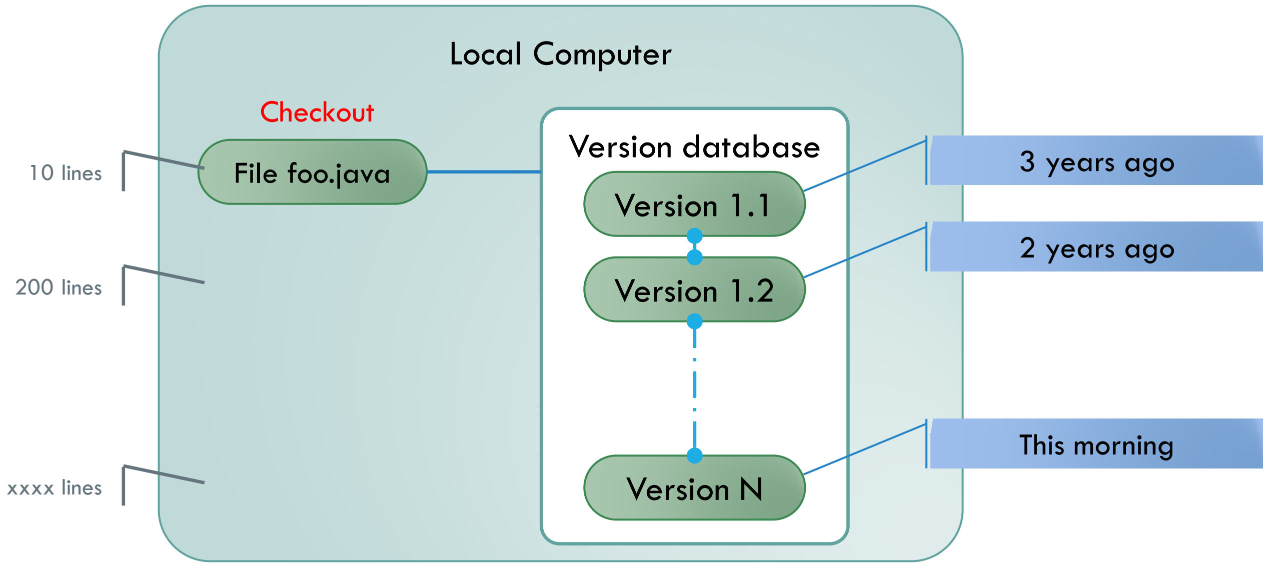
- **Conflict**: (occurs when different parties make changes to the same document, and the system is unable to reconcile the changes)
- **Fetch, Pull, Push**: (Copy revisions from one repository into another)
- **Repository**: (a data structure that stores metadata for a set of files or directory structure)
- **Tag**: (an important snapshot in time, consistent across many files)
- **Working copy**: (the local copy of files from a repository, at a specific time or revision)
- **Merge**: (or integration is an operation in which two sets of changes are applied to a file or set of files)
- **Head**: (the most recent commit, either to the trunk or to a branch)

2. VERSION CONTROL SYSTEMS (E.G., GIT)

Version control software:

- Apache Subversion
- Git
- Mercurial
- Perforce
- Concurrent Versions System
- Azure DevOps Server
- GNU Bazaar

2.1. LOCALIZED VCS



2.2. CENTRALIZED VCS

In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code.

- the server maintains the sole version history of the repo

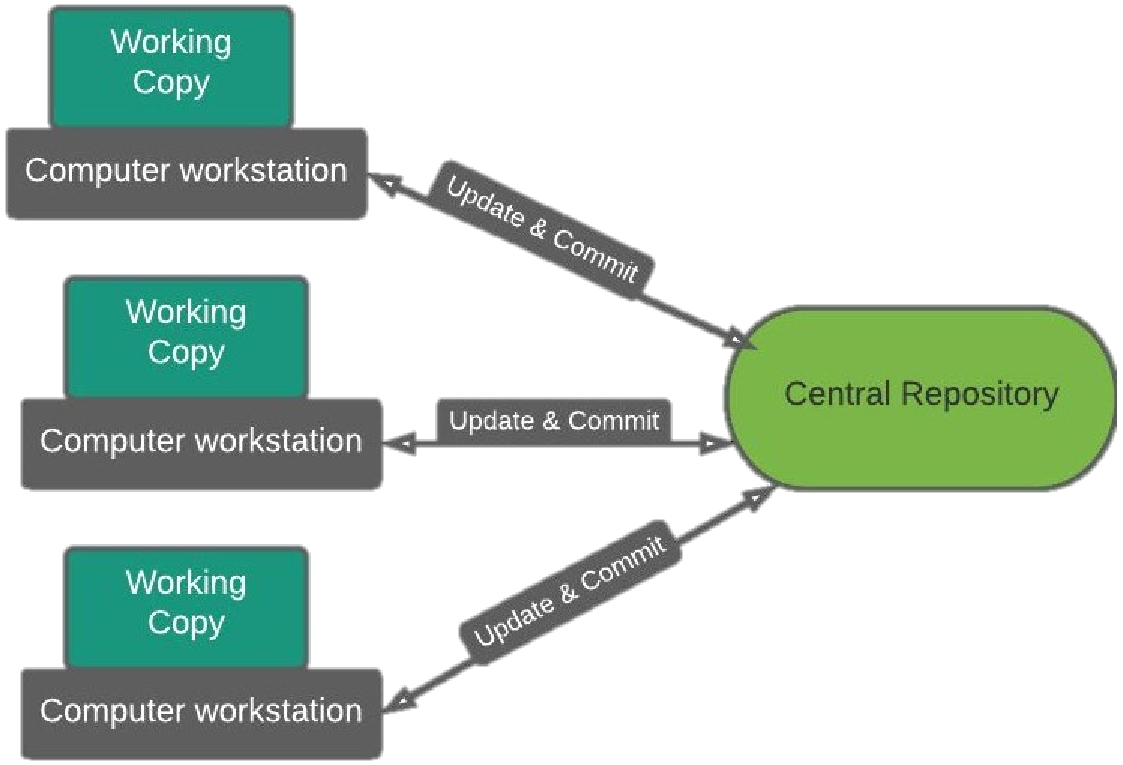
You make "checkouts" of it to your local copy

- you make local modifications
- your changes are not versioned

When you're done, you "check in" back to the server

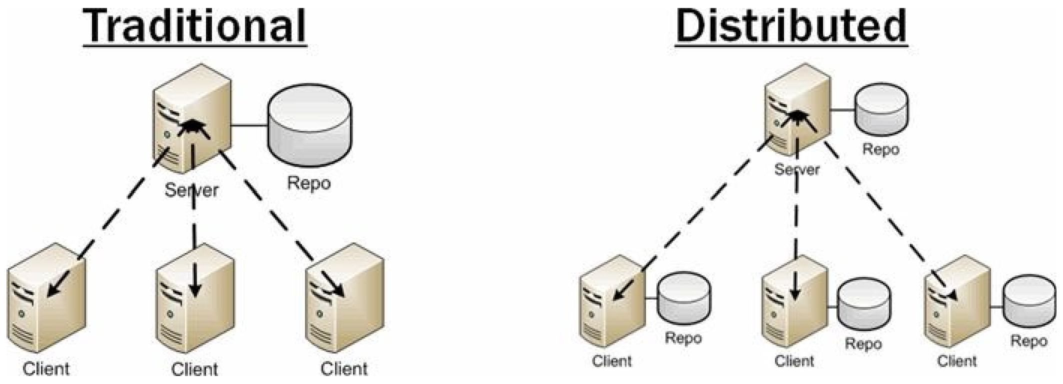
- your checkin increments the repo's version

2.2. CENTRALIZED VCS



2.3. DISTRIBUTED VERSION CONTROL SYSTEMS

In a distributed version control system each user has a complete local copy of a repository on his individual computer.



2.3. DISTRIBUTED VERSION CONTROL SYSTEMS

In git, mercurial, etc., you don't "checkout" from a central repo

- you "clone" it and "pull" changes from it

Your local repo is a complete copy of everything on the remote server

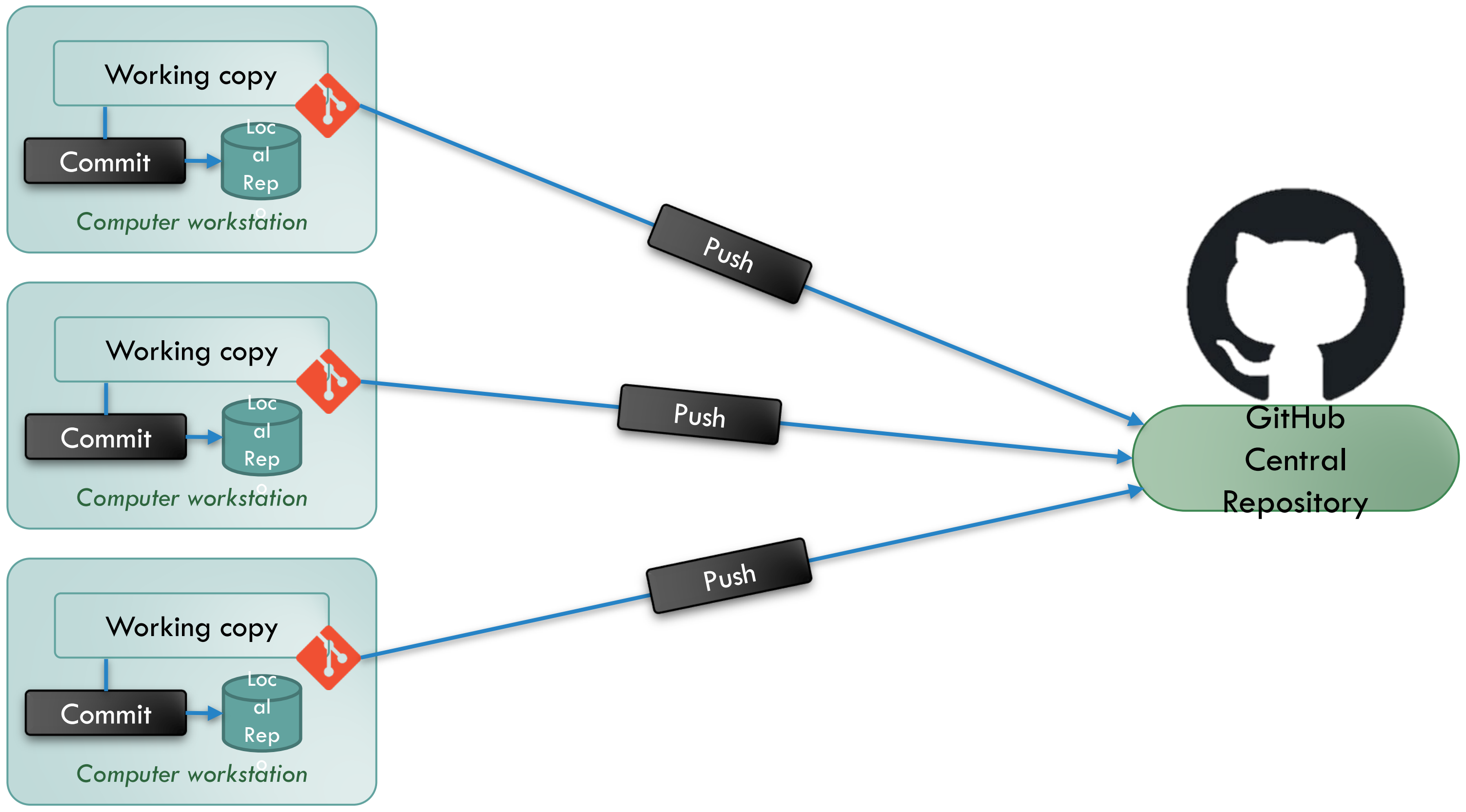
- yours is "just as good" as theirs

Many operations are local:

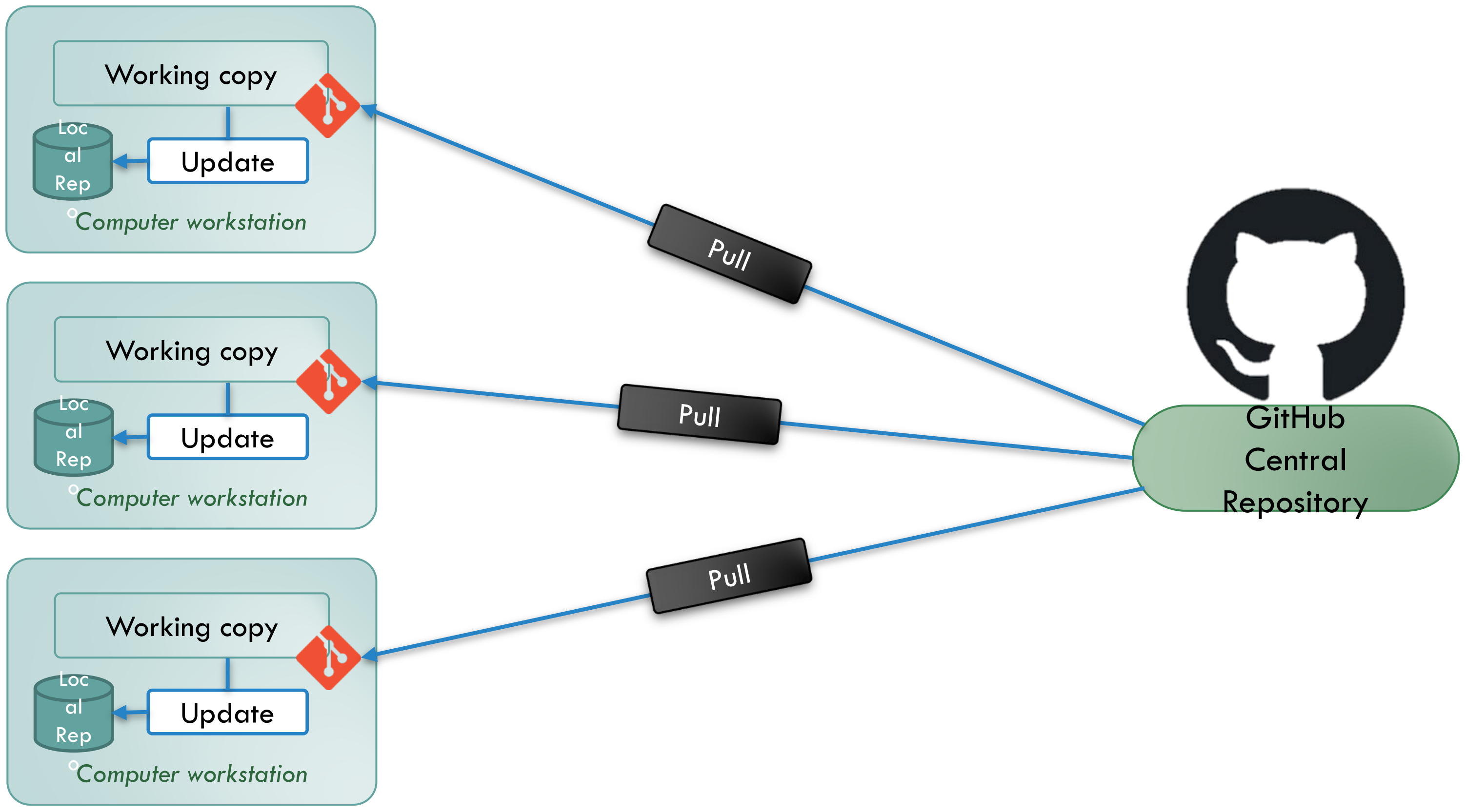
- check in/out from local repo
- commit changes to local repo
- local repo keeps version history

When you're ready, you can "push" changes back to server

2.3. DISTRIBUTED VERSION CONTROL SYSTEMS



2.3. DISTRIBUTED VERSION CONTROL SYSTEMS

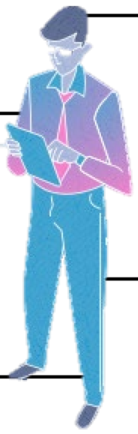


2.4. WHAT IS GIT?

Git is a distributed version control system

Git is a Tree History storage system

Git is content tracking management system



Ease
Simple to use tools & commands.
Cloud based remote repository.

Speed

- Support for non-linear development
- Fully distributed
- Able to handle large projects

2.5. GIT QUICK START

Created by Linus Torvalds, creator of Linux, in 2005

- Came out of Linux development community
- Designed to do version control on Linux kernel

Installing Git

Install Git Bash

Install via HomeBrew

Install via Package manager (yum, apt, snap etc)

2.6. LOCAL REPOSITORY SETUP

1. Set the name and email for Git to use when you commit:
   - ✓git config --global user.name “SOK Bopha"
   - ✓git config --global user.email bopha.sok@email.com
2. Create a directory
3. Initialize directory with
   - ✓git init
4. Create Readme.md file
   - ✓git add (Staging)
   - ✓git commit (Local commit)

2.7. REMOTE REPOSITORY

Create Remote repository on

- ✓GitHub, Gitlab, bitbucket etc.

Clone Repo to local

- ✓git clone URL

Local to Remote integration

- ✓cd to local repo
- ✓git remote add origin ssh://git@github.com/[username]/[repository-name].git
- ✓git push
- ✓git pull (to fetch latest changes)

3. CODE REVIEWS AND PAIR PROGRAMMING

Code reviews are methodical assessments of code designed to identify bugs, increase code quality, and help developers learn the source code.

The code review process is also an important part in spreading knowledge throughout an organization. For those reasons and more, 76% of developers who took the 2022 Global DevSecOps Survey said code reviews are "very valuable."

3.1. CODE REVIEW TECHNIQUES

There are 4 approaches to code review:

- **Pair programming** (two developers collaborating in real time — one writing code (the driver) and one reviewing code (the navigator))
- **Over-the-shoulder reviews** (two developers — the author and reviewer — team up in person or remotely through a shared screen)
- **Tool-assisted reviews** (automatically gather changed files to provide feedback and have conversations via comments, and incorporate things like static application security testing (SAST) to help identify and remediate vulnerabilities)
- **Email pass-around** (an author sends an email containing code changes to reviewers)

3.2. PAIR PROGRAMMING

Pair programming involves two developers collaborating in real time — one writing code (the driver) and one reviewing code (the navigator).

Team members share knowledge and can quickly overcome difficulties by working through ideas together and drawing on their expertise.

Switching roles

The programmers frequently switch roles so that both are actively involved and engaged.

Best practices

To ensure both programmers are contributing equally, you can:

- Regularly switch roles
- Create an environment where both programmers feel comfortable sharing ideas
- Be mindful of power dynamics and address them through open communication

3.2. PAIR PROGRAMMING

The benefits of pair programming

- Transfers knowledge
- Solves complex problems
- Increases morale
- Finds more bugs
- Can be conducted remotely

The drawbacks of pair programming

- Time-consuming
- Can be overused
- Difficult to measure

REFERENCES

https://www.geeksforgeeks.org/coding-standards-and-guidelines/

https://www.perforce.com/resources/qac/coding-standards

“Design patterns elements of reusable object-oriented software” by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995.

https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

https://about.gitlab.com/topics/version-control/what-is-code-review/

https://andreigridnev.medium.com/code-reviews-and-pair-programming-68a5ca8ba90c