Professional Workflow

Name

Universidad Panamericana

Presentation August 26, 2024

UNIVERSIDAD
PANAMERICANA

Contents

Why good practices?

Why good practices?

- **1. Maintainability**
  - This is crucial for long-term projects where multiple developers might be working on the same codebase over time.
- **2. Readability**
  - Clear and consistent coding standards make it easier for developers to read and understand each other's code.
- **3. Reusability**
  - This means that code can be reused in different parts of a project or even in different projects, saving time and effort.
- **4. Bug Reduction**
  - Identifying and fixing bugs early in the development process.
- **5. Performance**
  - This is particularly important in applications where performance is critical, such as real-time systems or high-traffic web services.

Why Good Practices?

- **6. Scalability**
  - Be easily extended with new features wiYout significant rework.
- **7. Security**
  - Secure data handling are crucial in preventing security breaches.
- **8. Documentation**
  - For future maintenance, debugging, and onboarding new developers.
- **9. Consistency**
  - It allows developers to switch between different parts of the codebase wiYout needing to adjust to different coding styles.
- **10. Professionalism**
  - It can enhance the reputation of a development team or company and build trust with clients and stakeholders.

10 commandments

1. **You shall prioritize Maintainability**
2. **You shall value Readability**
3. **You shall strive for Reusability**
4. **You shall reduce Bugs early**
5. **You shall optimize Performance**
6. **You shall ensure Scalability**
7. **You shall secure thy code**
8. **You shall document thoroughly**
9. **You shall maintain Consistency**
10. **You shall uphold Professionalism**

Use other programming languages

You shouldn't be opposed to learning programming languages.

- Latex : Documentation
- Java: PlantUML
- JSON: For automatization stuff
- Python: Unit Testing
- C: Pretty basic programming
- Cpp: Robust programming
- C#: Windows App

GOOD SOFTWARE PRODUCT CYCLE

Requirments

Define functionality, scope, goals. Try to cover any hole, basically is a contract to met.

https://doorstop.readthedocs.io/en/latest/

Software detailed document

Be specific, give more details about how met your requirements, which tools, libraries you will use.

UML

Visual aid and first code planning for your product. First design then code.

Planning

Split your work into task and estimate the amount of work. Divide and conquer.

Develop

The funniest part, just please don't start here. You must follow good software practices

https://code.visualstudio.com

Testing

Learn to think as an user, create corner cases, at least accomplish the minimum valuable product functionality.

Git tracking

Code review

Your code must follow coding standards, good practices. Also we may have some mistakes ever.

Fix Bugs

Sometimes after release a bug can come and attack you. Just squash it and solve it.

Requirements

Requirements

DOORS is a requirements management tool used by organizations to manage project requirements throughout the development lifecycle. It helps in capturing, analyzing, tracing, and maintaining changes to information to ensure a project's compliance with its initial requirements. This tool is particularly useful for managing complex projects, providing traceability, and improving collaboration and verification efforts. Requirements

SDD

Software Detailed Document (SDD)

The SDD serves as a guide for developers during the build phase and aids in maintaining consistency and understanding of the system's design principles and functionalities. SDD

UML

Unified Modeling Language with PlantUML (UML)

UML diagrams serve as excellent documentation tools that are useful throughout the system's lifecycle. They help new team members understand the system quickly and can also be valuable for maintenance and future upgrades. PlantUML

Abstract class as interface

- **Abstract Class in C++:** In C++, an abstract class is defined by declaring at least one pure virtual function, which is a function with no implementation (denoted by '= 0'). This makes the class abstract, meaning it cannot be instantiated directly.
- **Interface Concept:** In C++, abstract classes are often used to define interfaces. An interface in this context is a class that provides a set of virtual methods that derived classes must implement. This allows you to enforce a consistent API across different subclasses.
- **Pure Virtual Functions:** A pure virtual function in C++ is declared like this: 'virtual void functionName() = 0;'. Any class that inherits from an abstract class must provide an implementation for all pure virtual functions, or it will also be considered abstract.
- **Usage in C++:** Abstract classes are used in C++ to create a common interface for a group of related classes. For example, an abstract class 'Shape' could define a pure virtual function 'draw()', forcing all derived classes like 'Circle', 'Square', or 'Triangle' to implement their own version of 'draw()'.

Abstract Class as Interface in C++

- **Example:**

```
// Abstract base class
class Shape {
public:
  virtual void draw() = 0; // Pure virtual function
};

// Derived class
class Circle : public Shape {
public:
void draw() override {
  // Implementation for drawing a circle
  }
};
```

Enums to Avoid Magic Numbers

- **Magic Numbers:** Magic numbers are hard-coded values in your code that have no clear meaning or context. They make the code difficult to understand and maintain, as the purpose of these numbers is not immediately clear.
- **Problem with Magic Numbers:**
  - Reduce code readability.
  - Make the code error-prone, as changing a value requires finding and updating every instance.
  - Lack of context or explanation for what the number represents.
- **Solution: Using Enums:**
  - Enums (enumerations) provide a way to define a set of named integer constants.
  - They improve code readability and maintainability by replacing magic numbers with descriptive names.
  - Enums also allow for easier updates since you only need to change the value in one place.

Example: Using Enums in C++

```
// Avoiding magic numbers using enum
enum Direction {
  North = 0,
  East = 1,
  South = 2,
  West = 3
};

void move(Direction dir) {
  if (dir == North) {
      // Move north
  } else if (dir == East) {
      // Move east
  }
  // and so on...
}
```

- **Benefits:**
  - **Clarity:** The code becomes self-documenting, making it easier to understand the purpose of the values.
  - **Maintenance:** Changing the underlying value of an enum only requires updating the enum definition, not every occurrence in the code.
  - **Safety:** Enums provide type safety, reducing the likelihood of errors from using incorrect values.

SOLID Principles

- **SOLID:** A set of five design principles that promote better software architecture in object-oriented programming.
- **S - Single Responsibility Principle (SRP):**
  - A class should have only one reason to change.
- **O - Open/Closed Principle (OCP):**
  - Classes should be open for extension, but closed for modification.
- **L - Liskov Substitution Principle (LSP):**
  - Subclasses should be replaceable for their base classes without altering the program's behavior.
- **I - Interface Segregation Principle (ISP):**
  - Clients should not be forced to depend on interfaces they do not use.
- **D - Dependency Inversion Principle (DIP):**
  - Depend on abstractions, not on concrete implementations.

Course Activity: Practicing Critical Software Components

During this course, you will engage in hands-on practice to solidify your understanding of key software design principles and components. The activities include:

- **Design an Abstract Class in PlantUML:**
  - Create a PlantUML diagram for an abstract class that encapsulates common functionalities across all controllers.
- **Develop Base Classes:**
  - Write two base classes, which can be either realistic or mock implementations, to practice inheritance and abstraction.
- **Define Enums:**
  - Create enumerations for potential errors, sensor types, and processing steps, ensuring clarity and avoiding magic numbers.
- **Present Your Design:**
  - Showcase your PlantUML design on the projector, explaining your design decisions and how they align with SOLID principles.
- **Justify with SOLID:**
  - Ensure that your design choices adhere to SOLID principles, providing justification for each decision based on these foundational guidelines.

Planning

Planning

In Scrum, a popular agile framework used in software development, two fundamental concepts are "story" and "sprint." Here's a brief explanation of each:

- A user story is a brief description of a feature from the user's perspective, aimed at ensuring the team delivers value based on user needs. It's formatted as: "As a [user], I want [goal] so that [reason]."
- A sprint is a time-boxed period (usually 1-4 weeks) where a team completes a set work chunk to produce a shippable product increment, incorporating planning, development, and review.

Taiga IO

Develop

Coding Standard

Google coding standard

- Prevent using magic numbers instead use enums, or constantands or finally defines.

Linter vs Compiler

- Clang tidy is a linter is a tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs.
- A compiler translates code written in a high-level programming language into a lower-level language, typically machine code that a computer's processor can execute directly.

ESP IDF

- ESP-IDF is designed to make it straightforward to program ESP32 functionalities in C or C++,
- Offers advanced features, real-time capabilities, and robust system-level functions for efficient management of tasks and power.

More Visual studio code extensions

- vscode-Icons
- Bracket pair coloriser
- Bookmarks
- Cs 128 Clang-tidy
- Enumerator
- GitLens
- Meld diff
- Plant UML
- Todo tree
- Live Share

TPD

Test plan document (TPD)

A Test Plan Document (TPD) outlines the strategy, resources, scope, and timeline for testing activities within a software project. It serves as a blueprint that guides the testing process, detailing what needs to be tested, how the testing will be conducted, who will perform the tests, and the expected outcomes TPD

Testing

Black box vs Whitebox testing

- Black Box Testing: Tests the functionality of software without knowledge of its internal workings, focusing on input and output.
- White Box Testing: Examines the internal structure and workings of software, requiring knowledge of the code to ensure through testing of internal operations.

Embbeded Unit Testing Framework

- Unit Testing is a way to check your code behaves systematically.
- https://github.com/espressif/pytest-embedded
- Example can be seen in `FreeRTOSCourse/Projects/dac_cosine_wave/pytest_dac_cosine_wave.py`

Git

Git

Open Git Presentation