Introduction to Software Dev Notes

Marie Burer

16-1-24

Contents

1 Overview

Tools:

- Warp
- Eclipse
- Git

What is Warp?

- Existing code base
  - Not huge, but bigger than other class projects
  - Useful because it reflects real world, you won't join a team and write code from scratch
- Written by Professor Goddard for a research project called WARP
  - Initially written in Swift, then rewritten in Java for the course
  - Primary purpose is to develop programs for network communication in wireless sensor networks

Why Warp?

- Get used to reading and modifying others' codes
- Learn to refactor code

- Rich opportunity to add:
  - documentation
  - pre- and post-conditions
  - testing
  - new features
- Learn from Professor Goddard's experiences and mistakes.

2 Software Development Process Models

Beginner's Model: Code and Fix

- Conceptual Development
- Code and Fix
- Release Product

Software Development Life Cycle

- Conception
- Requirements gathering
- Design
- Coding and debugging

- Testing
- Release
- Maintenance
- Retirement

Software Development Model Categories

- Plan driven models:
  - Strict methodology
  - Clearly defined phases
  - Heavy weight
  - Works best for large contracts
- Agile development models:
  - incremental and cyclic
  - small, frequent releases
  - less documentation
  - Works well for start ups

Waterfall Process Model

- Conceptional Development
- Requirements Analysis

- Architectural Design
- Detailed Design
- Code and Debug
- System Testing
- Release and Maintenance

3 Code Construction

Who is the code for?

- The computer

  Must fulfill requirements and implement the design

- You and other programmers

  Must be readable and easy to understand

4 Debugging

Testing finds errors, debugging repairs them

5 Stuff I Didn't write down

6 Object Oriented Design Principles

6.1 Liskov Substitution Principle

The key of OCP: Abstraction and Polymorphism

Implemented by inheritance, how do we measure quality?

Example: Birds can fly, penguins cannot fly.

Throwing an error does not model "penguins cannot fly", it models "try, error"

Design by contract:

Advertised behavior of an object, requirements and promises

Derived class services should require no more, promise no less.

Is-A relationship must refer to the behavior of the class

Replaceability: Any code which can legally call a class's methods must be able to substitute any subclass without modification

6.2 Dependency Inversion Principle

High-level should not depend on low-level, both should depend on abstractions

Design to an inheritance not an implementation

Abstract classes/interfaces are not subject to change

6.3 The Founding Principles

Closely related, violating one means violating others

Keep them in mind at all times