Good Software

Lecture 6
GSL Peru 2014

What is Good Software?

- Low cost
- Good performance
  - Bug-free, efficient, meets its purpose
- Easy to code
  - Easy to understand, modular
- Easy to use
  - Clients are satisfied

Developer's Perspective

"High-quality software is not expensive. High-quality software is faster and cheaper to build and maintain than low-quality software, from initial development all the way through total cost of ownership."

-Capers Jones

Developer Perspective

CISQ (Consortium for IT Software Quality) Standards:

- Reliability
- Efficiency
- Security
- Maintainability

Developer Perspective: *Reliability*

Measures the level of risk and likelihood of potential application failures

- Compliance with object-oriented and structured programming practices
- Avoid software patterns that lead to unexpected behaviors
- Avoid Dirty programming

Developer Perspective: *Reliability*

- OOP
  - Clear data structure relations
  - Modular
- Quality Control
  - Functionality met, Bug-free, Easy to use
  - Unit Tests
  - Peer Reviews

Developer Perspective: *Efficiency*

Measure of source code efficiency and scalability

- Memory, network, disk space management
- Data access performance and management
- Coding practices (efficient algorithms)

Developer Perspective: *Efficiency*

- Design Patterns
  - General solution to a common, recurring problem
  - Program organization, common data structures, algorithms, computation
  - OOP: Shows the relations and interactions between unspecified classes and objects
- Optimization
  - Execution time, memory usage, disk space, power consumption, bandwidth
  - Design level - algorithm space/time efficiency
  - Avoid poor coding

Developer Perspective: *Efficiency*

- Code profiling
- Memory Usage Monitoring
- Load testing
- Application Verifier (Windows)

Developer Prospective: *Security*

Measure of potential security breaches due to poor coding and architectural practices

- Secure controls: access to system functions, access control to programs
- Programming practices: code-level error-exception handling
- Multi-layer design compliance

Developer Prospective: *Security*

- Static Code Analysis Tools
  - Flawfinder
  - Compiler warning - GCC, etc
  - Lint
- Dynamic Code Analysis Tools
  - valgrind
  - fsnoop
  - Application Verifier
- Penetration Testing

Developer Perspective: *Maintainability*

Notion of adaptability, portability, and transferability of code within a business

- Modularity, Understandability, Reusability, and Testability
- Source code file organization
- Architecture, program, and code-level documentation

Developer Perspective: *Maintainability*

- Good, clean code
  - Modular
  - Balance of comments and whitespace
- Object Oriented Programming/Modular Programming, ADTs
  - OOP/Modular allow code reuse
  - Abstract Data Types are models for data structures with similar behaviors
  - Using OOP/Modular and ADTs ease the understanding and evolution of code over time

Developer Perspective: *Maintainability* (contd.)

- Interfaces
  - Do not contain the implementation for functions
  - Contain functions that are common to different classes, but the classes define the implementation
- Relation between OOP, ADTs and Interfaces
  - Objects/Data structures: provide specific implementation to ADTs by **extending** ADTs
  - Abstract Data Types: provide specific behaviors but not full implementations by **implementing** interfaces
  - Interfaces: Contain the documentation matching the specific behaviors of the ADTs

Software Development Process

- Requirements
- Design
- Testing
- Peer Review

The Process:
Requirements

Know the product requirements before starting to create/build the product.

- Purpose or goal from customer's point of view
- Objectives
- Features

The Process:
Design

Create a design for the product before building.

Using **interfaces** leads to clean, maintainable code.

Understand the overall system architecture and potential bottlenecks.

The Process:
Testing

Create Unit Tests before writing the code.

- Create tests for individual modules (method, class, interface) based on requirements
- Use systematic approach: partition input space
- Regression testing: Update tests and code as bugs are found
- Automation: run and check test results without manual effort

The Process:
Peer Review

More eyes on your code, better the code!

- Buddy Checking - simple peer review
- Walkthrough - group review of a component
- Software Inspection - Inspect code for defects

Software Development Tools

- Source Control
- Continuous Integration
- Peer Review
- Developer Productivity Tools

Tools:
Source Control

- Maintain revisions of source code
- Easy to revert mistakes and keep an audit trail
- Examples
  - GIT, Subversion, Perforce, Starteam, etc.

Tools:
Continous Integration

- Automated builds and unit testing
- Part of XP and TDD
- Examples
  - Cruise Control, TeamCity, etc

Tools:
Peer Review

- Maintain Code Quality
- Examples
  - Code Collaborator, github, Crucible, etc

Tools:
Developer Productivity Tools

- Increase Coding Efficiencies
  - Refactoring Tools
    - ReSharper, IntelliJ IDEA, Eclipse,PyCharm, etc
- Increase Code Quality
  - Code Coverage Analysis
    - dotCover, JaCoCo, Emma, etc
  - Profilers
    - dotTrace, Netbeans, etc
  - Memory Leak Detectors
    - ANTS, Purify, JProfiler

Coding Standards

- Set of guideline to write code to
- Create maintainable code through standardized style
- Does not work if everyone does not adhere to it

- Hungarian notation, CamelCase

Examples

- OpenStack - https://wiki.openstack.org/wiki/CppCodingStandards
- http://www.possibility.com/Cpp/CppCodingStandard.html
- http://www.cs.northwestern.edu/academics/courses/311/html/coding-std.html
- Java - Sun's Java Coding Standards

Coding Standard Enforcement

Continuous Integration Plus

- C/C++ - lint - static code analysis
- C# - StyleCop, FxCop
- Java - CheckStyle
- Python - pylint, pyflake, pychecker

Don’t Repeat Yourself (DRY)

```
vegetables = ['asparagus', 'broccoli', 'cabbage']

print vegetables[0], 'is a vegetable'
print vegetables[2], 'is a vegetable'

def print_veg(index):
    print vegetables[index], 'is a vegetable'
print_veg(0)
print_veg(2)
```