Implementation

(Chapter 7)

Characteristics of a good Implementation

- **Readability**: code can be easily read and understood by other programmers.
- **Maintainability**: code can be easily modified and maintained.
- **Performance**: code performs as fast as possible.
- **Traceability**: all elements of code should correspond to a design element
- **Correctness**: it should perform as intended, with respect to requirements and detailed design.
- **Completeness**: it meets **all** system requirements.

Tradeoffs and interactions of characteristics

- Readability usually helps maintainability.
- Readability and maintainability usually help achieve correctness
  - how? debugging is much easier.
- Performance optimizations often reduce readability and maintainability.

How to achieve the desired implementation characteristics

- Readability and maintainability
  - Programming style and coding guidelines
  - Using comments well
  - Refactoring
- Correctness
  - Testing and debugging
- Performance
  - Optimization

Programming style and coding guidelines

- Naming
  - Good names contribute significantly to improving readability.
  - Well chosen names convey the intent of the element
  - Poorly chosen names are misleading and confusing
    - often indicate programmer does not understand the code or that the element is poorly designed.
  - File name should correspond to elements it contains
- Indentation
  - Use indentation to reflect the structure of the code
- Function size
  - Large functions are more error prone (and less cohesive)

Comments

- Should be used to enhance understanding of code
  - Good example: explaining the interface of a class or function
- Problems:
  - When they distract from the code (clutter)
  - When they are wrong or misleading
- Examples of poor uses of comments
  - Commenting out entire sections of code
    - may not be clear it’s commented out
    - why is it there?
  - Comments that explain the code
    - usually a cover up for poorly written code
  - Commenting out output statements used for debugging
  - Indicating when code was changed by who for what reason
    - This info can be found using version control system

Debugging

- Fixing errors in the code
  - especially run-time/logic errors
- Process:
  1. Reproduce the error
     - Write a test case that demonstrates the error
  2. Find the section of code that leads to the error
     - See next slide
  3. Correct the code
     - Don’t do this first! Don’t guess!
  4. Verify the fix
     - Re-run the test case and make sure you get no error

Debugging

Debugging methods:

- Temporary output statements inserted into code:
  - view values of variables
  - analyze control flow
- Interactive debuggers
  - Tool used to view variables, step through the code, insert breakpoints
  - Sometimes have a steep learning curve
- Profilers
  - Tool that gives statistics about code, or memory while code is executing, or other metrics

Performance Optimization

- Improving performance requires changes to code that often make it less readable and maintainable.
- Many programmers worry about performance too early.
  - Instead you should write readable code first and then add performance improvements later, as needed.
- How to optimize:
  - Use a profiler to determine how much time is spent on each part of the program
  - first get a baseline, find the problematic areas
  - after code is modified, run profiler again and compare to baseline.

Implementation issues

- Aspects of implementation that are important to software engineering but not covered in programming textbooks
  - **Configuration management**: managing the different versions of each software component (the source code).
  - **Open source development**: when the source code of the system is publicly available.

Configuration management

- Potential problems of team development
  - Interference: Changes made by one programmer could overwrite a change previously made by another.
  - Redo good work: Programmers accessing out-of-date versions could re-implement work already done.
  - Can't undo bad work: Figuring out how to undo problems introduced into a previously functioning system.
- Configuration management: Process of managing a changing software system, so all developers can
  - access code and documentation in a controlled way
  - find out what changes have been made
  - compile and link components to create the system.

Fundamental configuration management activities

- Version management
  - track different versions of the files in the program
  - coordinate work of multiple developers.
- System integration
  - define which versions of each component and/or file are used for a given version of the overall system.
  - then builds system automatically
- Problem tracking
  - allows users to report and track bugs.
  - allows developers to track progress on fixing bugs.

Configuration management tools

- Integrated tools: all three components in one
  - tools share same interface, can share information
  - ClearCase
- Version management
  - CMVC, CVS, subversion, git, mercurial.
- System integration (build tools)
  - make (unix), Apache Ant, or built into IDE
- Problem tracking
  - bugzilla
  - any database

Open source development

- The source code of the system is publicly available
- Volunteers are invited to participate in the development process (may be users).
- Some open source projects:
  - Linux, Apache web server, Java
  - Eclipse, FireFox, Thunderbird, Open Office
- Issues for the developer:
  - Should an open source approach be used for the software's development?
  - Should the system being developed (re)use open source software components?

Open source development

- How to make money developing open source products?
  - Development is cheaper: volunteer labor.
  - The company can sell support services
  - Software must have wide appeal
- Re-using open source software in software products:
  - These components are generally free.
  - These components are generally well-tested.
  - There may be licensing issues. . .

Open source licenses

- GNU General Public License (GPL).
  - reciprocal
  - if you re-use this open source software in your software then you must make your software open source.
- GNU Lesser General Public License (LGPL)
  - you can write components that link to open source code without having to publish the source of these components.
- Berkley Standard Distribution (BSD) License.
  - non-reciprocal
  - not obliged to re-publish any changes or modifications made to open source code.
  - you may include the code in proprietary systems that are sold.