Midterm Review

Christopher Simpkins
chris.simpkins@gatech.edu

Version Control Systems

- Records changes to files over time
- Allows you to recover older versions of a file (effectively “undoing” recent chagnes)
- Many people already do this
  - manually by saving copies of files with version information embedded in the file name, or
  - automatically using a commercial solutions such as Apples Time Machine.

Neither of these approaches is sufficiently powerful for professional software development.

Version Control Systems

Architectures

- Local VCS (RCS)
- Centralized client-server VCS (CVS, Subversion)
- Distributed VCS (Git, Hg, Darcs)

Locking Models

- Pessimistic locking (most commercial VCSes)
- Optimistic locking (Most open-source VCSes - CVS, SVN, Git, Hg)

Pro Java

- The classpath
- Separating source and compiler output
- Project directory layout
- Packages
- Jar files

Agile Software Processes

Software development lifecycle

- Risk Management
- Sequence of activities
- Conceptual models: waterfall, spiral/iterative

Agile software development

- Individuals and Interactions over process and tools
- Working software over comprehensive documentation (design as you go – no big up front design)
- Customer Collaboration over contract negotiation
- Responding to Change over following a plan

Testing

- Test writing techniques
  - Boundary conditions (on boundary, and one off)
  - Equivalence partitions (one case per)
  - “Plausible” faults (specific values)
- Black box and white box tests
- Test-driven development
  - **First Law** You may not write production code until you have written a failing unit test.
  - **Second Law** You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
  - **Third Law** You may not write more production code than is sufficient to pass the currently failing test.
- Testing F.I.R.S.T. rules: tests should be fast, indepedent, repeatable, self-validating, and timely

Testing

F.I.R.S.T rules. Tests should be

- Fast to write, fast to run,
- Indepedent of other tests,
- Repeatable in different environments,
- Self-validating with asserts instead of reports to be interpreted manually, and
- Timely, written along with the code they test.

Clean Code

What is “clean code?”

- Elegant and efficient. – Bjarne Stroustrup
- Simple and direct. Readable. – Grady Booch
- Understandable by others, tested, literate. – Dave Thomas
- Code works pretty much as expected. Beatuful code looks like the language was made for the problem. – Ward Cunningham

Why do we care abou clean code?

- Messes are costly. Quick and dirty to get it done ends up not getting it done and you will not enjoy it. It’s lose-lose!
- We are professionals who care about our craft.

The Boy Scout Rule

Clean Names

- Names should mean something
- Names should reveal intent
- Name should not disinform
- Names should make distinctions
- Avoid encodings
- Avoid mental mapping. We're all smart. Smart coders make things clear.
  - So simple only a genius could have thought of it. – Einstein
  - Simplicity does not precede complexity but follows it. – Perlis
- Use nouns or noun phrases for class names.
- Use verbs or verb phrases for method names.
- Don't use puns or jokes in names.
- Use one word per concept.
- Use CS terms in names.
- Use problem domain terms in names.

Clean Functions

Functions Should be Small and Do one Thing Only

- The first rule of functions: functions should be small.
- The second rule of functions: functions should be small.
- One level of abstraction per function.
  - A function that implements a higher-level algorithm should call helper functions to execute the steps of the algorithm.
- Write code using the stepdown rule.
  - Code should read like a narrative from top to bottom.
  - Read a higher level function to get the big picture, the functions below it to get the details.
- The fewer the function parameters, the better
- Avoid side-effects
- Use return values rather than output parameters
- Prefer exceptions to error codes

Clean Comments

Sometimes comments are useful.

- Legal comments (copyright notices, licenses)
- Informative comments
- Explanation of intent
- Clarifications
- Warnings
- Todos
- Amplification
- Javadocs for public APIs

Most comments are bad.

- Redundancies
- Misleading
- Noise

Remember: comments make up for lack of expressivity in a programming language. You shouldn't need many, and you certainly don't need long comments.

Clean Formatting

The purpose of formatting is to facilitate communication. The formatting of code conveys information to the reader.
Vertical formatting

- Newspaper metaphor
- Vertical openness between concepts
- Vertical density
- Vertical distance
- Vertical ordering

Unified Modeling Langauge (UML)

A standardized diagrammatic language for communicating OO designs in a language-independent way. Very rich, but for now focus on:

- use cases,
- domain model (classes and associations),
- packages, and
- sequence digrams.

Clean Classes

- Data abstraction
- Data structure classes versus object-oriented classes
- Data/Object Anti-symmetry
  - Procedural code (code using data structures) makes it easy to add new functions without changing the existing data structures. OO code, on the other hand, makes it easy to add new classes without changing existing functions.
  - Procedural code makes it hard to add new data structures because all the functions must change. OO code makes it hard to add new functions because all the classes must change.
- Law of Demeter
- Single responsibility principle
- Open closed principle

Object-Oriented Design Principles

- **S**ingle Responsibility Principle: a class should have only one reason to change
- **O**pen Closed Principle: a system should be open for extension, closed for modification
- **L**iskov Substitution Principle: subtypes should be substitutable for supertypes
- **I**nterface Segregation Principle: clients should not be forced to depend on methods they don't use (break large interfaces into smaller, more focues ones)
- **D**ependency Inversion Principle: high level modules should depend on abstractions of lower level modules (not their details) so that lower-level modules can be swapped easily