# Getting Code Right

## PSU CS 300 Lecture 5-2

Bart Massey
Assoc Prof Computer Science
Portland State University
<bart@cs.pdx.edu>

## Principles of Coding

- **Style**
- **Formatting**
- **Correctness**

# Single-Entry Control Structures

- “Turing-complete”: can write anything
- Dynamic structure follows static structure = easy to read
- *vs* Single-Exit

## GOTO Considered

- GOTO considered harmful (Dijkstra) ***except***
  - natural labeling (*i.e.*, state machine)
  - loop exit *ala* C `break` statement
  - well-labeled unusual situation
  - ill-endowed language (*e.g.*, assembly)

## Formatting

- You already know how to do this
  - follow a consistent style
  - use plenty of white space
  - one statement per line
- Follow standards
  - cultural conventions, org rules
  - style of preceding programmers

## Naming

- Names can be
  - too short or long
  - insufficiently idiomatic
  - too clever
  - misspelled or ambiguous
- Name to avoid commenting

## Commenting

- Comments give *commentary*
- Comments are mandatory
  - at cleverness
  - top of any non-trivial module
  - any complex control flow
  - to cite references
- Comments *vs* documenting

# Assertions

- Assertions are
  - comments
  - debugging aids
  - compiler hints
- Retain forever if possible

## Cleverness

- Never ever be clever!
  - always choose the simplest way
  - comment where there is the slightest doubt
- Code should be *best translation* of detailed design
- See `http://www.ioccc.org/` for amazing counterexamples

## Optimization

- Optimize design, not code!
  - Massey/Packard 2x Rule
- You cannot predict what code will be slow
  - modern compilers are too clever
  - modern hardware is too complex
  - you do not understand your design well enough

## Tuning

- If you *must* tune code
  - comment it thoroughly!
  - retain and maintain unoptimized version
- Profiling is your friend
- Do tuning last

## Portability

- **Avoid the undefined: at least be cross-version**
- **Always choose clean over portable (initially)**
- **Modules, not conditions**
- **No gratuitous portability**

## Instrumentation

- Make state accessible
- Keep statistics
- Use reporting mechanisms that are
  - unobtrusive
  - usable

## Code Management

- Crucial modern advance
- Many types of tool
  - revision control
  - build management
  - code browsing and visualization
  - defect reporting and tracking

## Code Browsers

- Improved view of code
- Features include
  - *"cross-referencing"* variable/function use/def
  - *"pretty-printing"* or formatting
  - *abstraction* of code views
- Most common in OOP (why?)
- Modern way is IDE

## Defect Tracking

- Usually maintenance-phase
- Record defect information
- Allocate resources to repair
- Largely custom or integrated
- Things with names like “BugTraq”, Bugzilla

# Readable, Maintainable Code

- Good software development techniques produce code that is
  - simple
  - readable
- Combined with good maintenance techniques, this leads to long and successful product lifetimes