Software Quality Guidelines
Prof. Arthur P. Goldberg
Owner: APG, Version: 1.2.3 1/04\5

Application: Writing programming assignments in my Internet and Intranet Applications and
Protocols course.

These are some guidelines for writing good code. Follow them, and you’ll write great code.

Cite: Based on Tom Gelb’s *Software Inspection*, 1993, pp. 423-427.

**Semantics** The code should implement the specification, as defined in the assignment.

**Simplicity** The code should accomplish its task (implement the semantics) as simply as possible, so it can
be easily understood.

**Confined** The code should confine itself to the semantics, and not implement other functionality.

**Perform** The code should implement the specification with an efficient algorithm. The code should
perform well. Performance requirements may include: no busy waiting, efficient memory use, and good
scaling.

**Robust** The code should handle errors well. It should try to recover from system errors. Unrecoverable
system errors and input errors should produce comprehensible error messages.

**Non-repetitive** Basic concepts (in code or comments) should be stated only once. Subsequent use of the
concept should refer to the initial statement.

**Well Commented** Comments in the code should accurately, thoroughly, clearly and concisely describe
the code.

**Layout** The code layout (indentation, spacing, variable naming, etc.) should help make the code easy to
understand.

**Symbolic** The code should use symbolic constants, not hard-coded values.

**Resourceful** The code should use data structures and algorithms available in the language or its libraries
when appropriate.

**Tested** The code should be thoroughly tested. Test cases should be packaged with the code.

**Usable** The code’s interface(s) (which may be function calls, command line operations, a GUI or a
combination of the above) meets the needs of the target users.

**Paradigm** The code should employ its programming language(s) in their recommended paradigms.

**Some Comments**

Guidelines **Semantics**, **Simplicity**, **Confined** and **Tested** are concerned with software correctness. If
informal correctness proofs are appropriate, they should appear in the comments.

Guideline **Perform** is fairly vague, but difficult to make more precise. I’d be interested in suggestions for
improving it. Guideline **Perform** relates to **Resourceful**, because reusing good code in libraries can help
make efficient algorithms.

Guidelines **Non-repetitive** and **Symbolic** recommend non-redundant expression of ideas. Redundant
ideas are bad because they must be modified in multiple places. This can be difficult or impossible when
the multiple places become widely separated.