Stat 650/750 Programming Rubric

Christopher Genovese and Alex Reinhart

9 Aug 2016

Assessment Areas

There are four assessment areas, roughly aligned with the main themes of the course.

Practices This assessment area focuses on coding style and code readability. The criteria are described in terms of the `Programming Practices Checklist`, which is available on Blackboard, and in terms of the red flags described below.

Design This assessment area focuses on the conceptual structure of the code and the interactions among the pieces of the system. The criteria are described in terms of the `Code Design Checklist`, which is available on Blackboard, and in terms of the red flags described below.

Representations, Algorithms, and Data Structures This assessment area focuses on the approach to solving a problem: how the problem is represented conceptually, the choice and implementation of algorithms, and the choice and implementation of data structures.

Correctness, Elegance, Performance This assessment area focuses on the practical effectiveness of the solution. The solution should be correct, and whenever possible, elegant. While optimal performance is not necessary in general, some attention to performance is warranted.

Within each assessment area, there are two quality levels: Mastered, and Not Yet Mastered. The criteria for each level are described for each assessment area in the following sections.

Practices

- Mastered. Ideal features are generally in place; very few (if any) red flags are present.
- Not Yet Mastered. Ideal features are inconsistently implemented, and one or more red flags are present.

Practices Red Flags

- Inconsistent, misleading, or easily confusable names
- Inconsistent indenting and formatting of code blocks
- Inconsistent use of horizontal and vertical space
- Use of global variables without very good reason
- Use of “magic numbers” (hard-coded numeric constants) in code
- Function and class names that are neither meaningful, concrete, nor descriptive
- Documentation that describes obvious or trivial facts or approach that can be directly inferred from the code
- Documentation that is disproportionate to the content of the code and hard to maintain accurately.

Design

- Mastered. Ideal features are generally in place; very few (if any) red flags are present.
- Not Yet Mastered. Ideal features are inconsistently implemented, and one or more red flags are present.

Design Red Flags

- Long functions that perform many tasks
- Classes that expose internal state and representation
- Excessively nested conditionals and loops or long conditional chains
- Repeated blocks of identical (or nearly identical) code
- Strong coupling between different components (functions, classes, etc.)
- Knowledge shared with components that do not need it
- Assumptions about the implementations of classes or functions made by their users

Representations, Algorithms, and Data Structures

- Mastered.
  - The representation of the problem makes it easier to craft a solution and an effective design.
  - The algorithm is appropriately chosen for the problem at hand with a theoretical performance that is feasible for a realistic range of input sizes.
  - The data structure is appropriate to the problem with space that scales effectively for a realistic range of input sizes.
- Not Yet Mastered.
  - The representation of the problem omits or obscures important features necessary for the solution.
  - The algorithm is grossly inefficient or incorrect.
  - The data structure chosen is insufficient for the problem at hand or introduces excessive inefficiencies in either algorithm or space use.

Correctness, Elegance, and Performance

- Mastered. All tests and sample inputs yield the correct result. The approach is elegant and is able to handle realistically large inputs in a manageable time. (Note that in the previous assessment the efficiency considered was theoretical; here it is practical.)
- Not Yet Mastered. Some tests fail, or sample inputs yield incorrect results in core cases. The approach is substantially more complex or case-driven than is necessary. Runs on moderately large inputs try one's patience.