Rose-Hulman Institute of Technology

Settlers of C#tan

Milestone 3

Liz Hines * Matt Moore * Sam Kim

April 25, 2012

Contents

1 Metrics

For the remainder of the project we will track the following metrics:

- Maintainability Index
- Cyclomatic Complexity
- Depth of Inheritance Hierarchy
- Class Coupling
- Source Lines of Code

The definition, process for measuring, how each metric will be used, and why we plan to track it are detailed in each metrics respective section below.

1.1 Maintainability Index

1.1.1 Definition

The maintainability index is an index value between 0 and 100 that represents the relative ease of maintaining the code.

1.1.2 Measurement Process

This metric is automatically calculated using the built in code analytics suite in Visual Studio 2010 Ultimate Edition. The formula for this indirect measure is:

$$MAX(0,(171-5.2*ln(HalsteadVolume)-0.23*(CyclomaticComplexity)-16.2*ln(SLOC))*100/171)$$

1.1.3 Analysis Plan

We intend to track this metric so that we can make sure that our code base is acceptably maintainable. If the analysis shows that a section of code has become unmaintainable we will refactor.

1.2 Cyclomatic Complexity

1.2.1 Definition

Cyclomatic complexity is a software metric that directly measures the number of linearly independent paths through a program's source code.

1.2.2 Measurement Process

This metric is automatically calculated using the built in code analytics suite in Visual Studio 2010 Ultimate Edition.

1.2.3 Analysis Plan

We intend to track this metric so that we can make sure that our code base has an acceptable level of test coverage. If the analysis shows that the cyclomatic complexity is high enough that there is insufficient code coverage we will refactor.

1.3 Depth of Inheritance Hierarchy

1.3.1 Definition

Indicates the number of class definitions that extend to the root of the class hierarchy.

1.3.2 Measurement Process

This metric is automatically calculated using the built in code analytics suite in Visual Studio 2010 Ultimate Edition.

1.3.3 Analysis Plan

We intend to track this metric so that we can make sure that it is clear where methods and fields are defined or/and redefined. If the analysis shows that the depth of inheritance hierarchy has reached a point where it is difficult to determine where methods and fields are defined or/and redefined we will refactor.

1.4 Class Coupling

1.4.1 Definition

Measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration.

1.4.2 Measurement Process

This metric is automatically calculated using the built in code analytics suite in Visual Studio 2010 Ultimate Edition.

1.4.3 Analysis Plan

We intend to track this metric so that we can make sure that we are designing software with high cohesion and low coupling. If the class coupling metric indicates that that our classes have either low cohesion or high coupling we will refactor.

1.5 Source Lines of Code

1.5.1 Definition

Indicates the approximate number of lines in the code. A line is counted if it is an executable line or a data definition.

1.5.2 Measurement Process

This metric is automatically calculated using the built in code analytics suite in Visual Studio 2010 Ultimate Edition.

1.5.3 Analysis Plan

We intend to track this metric so that we can make sure that we are designing methods and classes that do not attempt to do to much. If the number of source lines for a class or method is particularly high we will investigate to see if the functionality can be abstracted better and if so we will refactor.

2 Revision History

Created 4/25/12 by Elizabeth Hines, Sam Kim, Matt Moore