Michael Kinsey,

Carlos Albaladejo,

and Daniel Stoeger

December 10, 2015

uCode Documentation

1. Installation

To run the application, simply click on the included executable application titled ‘uCode’ in the build directory. For instructions on using the product, see the User Manual Section. In order to build or recompile the project, the Qt Creator development environment should be used. Included with the source code is a ‘.pro’ project file that can be imported into Qt Creator to set up the project properly. One of the advantages of C++ and the Qt library is platform independence. The project can be compiled to run on nearly any operating system. For more information about modifying the source code, consult the Developer’s Guide.

1. User Manual

The uCode application contains a simple user interface. On the righthand side, the class panel is shown. Here, all of the information about a given class can be added.

**Working with classes**

A user may select a class type, ‘Base’, ‘Child’ or ‘Interface’ using the top three buttons. If a class inherits from another class, simply add the class name in the Base class box. uCode utilizes a grid layout for the diagram. a class’ position within the diagram can be specified with the X and Y text boxes. The default position is 0, 0. Methods take the form: (+/-) methodName : returnType. The default access modifier is public, and will be assigned if none is given. Attributes take a similar form: (+/-) attribute : type. Clicking the Create button will add the class to the diagram. If a base class is specified, and the corresponding class exists in the diagram, an Inheritance arrow will be drawn automatically. If a class is clicked in the diagram, it will become selected and its information will be loaded into the class panel. Attributes can then be edited and saved, using the Update button.

**Generating Code**

To generate source code from the diagram, choose “Code->Generate” from the top menu bar. A language and destination folder may then be selected.

1. Developer’s Guide

The uCode application leverages certain design patterns to make the code base more maintainable, testable, and extensible. When contributing to the application, these patterns should be properly incorporated. Style guidelines are included with the source code, and should also be observed.

**Adding a Language**

The most common addition to the application would be the incorporation of a new language for code generation. The application utilizes the strategy pattern to encapsulate all of the logic for a particular language in one class. A family of language patterns are defined, and a user can select the desired language at runtime. The visitor pattern is also used in the application. This pattern takes a language strategy and can then apply it to a group of objects. This pattern provides insulation between the object structure and the application logic that is applied to the structure. Furthermore, this pattern eases the creation of a new strategy. A developer with relatively little knowledge of the inner workings of the rest of the program can easily develop a new strategy based on the existing strategies.

**Changes to the UI**

User interface changes are very common. Changes can be made to reflect changes in design styles, but also to make an application more intuitive to users. uCode is well equipped to handle such changes. In fact, the application can function completely independently from the user interface, making the application very maintainable. The facade pattern facilitates this by simplifying common interactions between the application logic and the interface. The user interface could be replaced with a separate one, such as for mobile devices, and the application logic would not have to be modified at all.

**Adding new features**

The uCode application has been engineered to be very maintainable by leveraging the patterns mentioned above. Another design pattern that enables this is the factory. If one wanted to add a new feature for example creating a referenced parent class if it does not already exist in the diagram, it would be simple to do using the class factory. Another example would be dynamically switching class types, say from a base class to a child class if a parent is added. The factory handles all of the creation of class objects, and can decide the object’s type at runtime. The basic object oriented concepts are well established, so the risk of having to modify the interface is relatively low, making the factory an even better choice. Another pattern that increases maintainability is the composite pattern. In short, a uChildClass object contains references to any parent classes (which shares the same interface, uInheritable) making it a composite structure. The Composite pattern is advantageous because groups of objects (the composite) can be interacted with in the same manner that primitive objects are. This simplifies interaction with the object structure making the code more testable and maintainable.

**Summary**

Design patterns incur a certain overhead within the development process. That is, they do not represent the path of least resistance to fulfilling a single requirement. Common problems result from taking the path of least resistance, such as code rigidity, spaghetti code, and redundant/inefficient logic. Design patterns exist to alleviate these issues. With proper planning in the design phase, and a small amount of extra code in the implementation phase, an application can become testable, extensible, and maintainable for the rest of the code base’s lifespan. uCode was developed this way, and with proper care and nurturing, it can grow into a fully fledged application that provides value to developers everywhere.

2-4 page document describing how to install the app as well as a professionally done user user manual (short – how to use the app), and lastly, an explanation of how to maintain and extend the app by describing how to add to the functionality using the design patterns.