KWIC Analysis

IT 372 Software Maintenance Tim Roush

### Criteria KWICv1

In KWICv1 I implemented a non-clean code approach using code styles we started with in Java I. The entire solution is in a single file and is not very abstract. There is little separation of concerns other than the separation of the different methods. The criteria was to just get the result as quickly as possible without concern for the future, maintenance, or updating.

### Criteria KWICv2

In KWICv2 I went with a very modular approach. The idea was to make an implementation that would be versatile and in doing so be modularized. It parses a command line string for parameters to determine how to proceed. An MVC framework was implemented to separate high level concerns. The input and output methods are implemented using a strategy pattern to allow for expanding to include other strategies in the future.

### Which is more resilient?

KWICv2 is by far more resilient and future-proof. Concerns are separated in a way that allows for following the code and identifying exact components responsible for each task. Because a Strategy pattern was used, it is possible to use other forms of input or output other than the prescribed methods mentioned in the problem. If this code needs to have functionality added or modified, it is possible to extend particular components and add such upgrades. It is also much easier and possible to test this version due to the level of abstraction and modularity.

### Change to Database KWICv1

In KWICv1, the original source would have to be modified to allow for database interaction. This is due to it being hard-coded to only interact with a specified input file. This would be a violation of the open-closed principal.

### Change to Database KWICv2

In KWICv2 the input is determined by a Strategy pattern using the iInput interface. So passing any iInterface object designed to work with a database to the model class will make this interaction possible.

### Change to GUI

KWICv1 has the same issue with switching to a GUI interface. The hard-coding prevents this switch without modifying the original source. KWICv2 also has the same solution. A strategy is used making it possible to pass any iOutput object to the controller to add a new user interface.

### Other Possible Changes

Possible other changes might include changing the action performed on the data, which in the current case is a circular shift. I however did not plan for that and both will have issues with this change. However, the KWICv2 will be easier to modify due to it’s modularity and abstract classes.

### Which is Easier to Understand?

One could argue that KWICv1 is easier to understand due to the code being all in a single location. However, it is a long chunk of code that does everything. I would say that KWICv2 is easier to understand but I also have in-depth knowledge of the classes and structure.

### Which is Better Performing?

KWICv1 is possibly better performing as there are less stack operations being performed. There is also less decision making due to the modularity of KWICv2 and the need to determine which output, or input to use.

### Information Hiding

Information hiding is the act of making a class or method so abstract and modularized that a parent or calling class or method has no idea what is being done or how it’s being done. All it cares about is getting the right results after passing certain data in. KWICv2 uses a heavy amount of information hiding in the Strategy pattern implementations. The model has no idea what is happening within an iInput strategy object. It just cares that is gets data out as a result.