BlueJ Text Editor Specifications
by
Holmes, Benjamin
Jain, Ayushi
Magee, Kylyn

Submitted in partial fulfillment
of the requirements for
CS 7140 Course

Department of Computer Science
Wright State University
15 October 2013
ABSTRACT

There exists a large base of prospective programmers, who, while interested in object-oriented design, are new to programming in general, and therefore ignorant of good object-oriented habits and practices. One major problem with their developing skills is that some of the abstractions and concepts involved in object oriented design - encapsulation, abstraction, polymorphism, and inheritance - can be fairly arcane and difficult to grasp for someone unused to programming. BlueJ will help ease this learning curve; the text editor and supporting graphics features will help users visualize and correctly format good OOD.

This document is designed to concretely specify, for developers, the concepts put forth in the requirements document. It will do so by refining the specifications and enumerating the ambiguities found in the requirements document. This document focuses on the specifications of the text editor portion of the BlueJ application. It will discuss the features of the text editor and some test cases that will check the performance of the functionalities. It will also discuss the mathematical and logical components and describe the specifications for buffers, cursors, view-windows, cut-paste-containers, keyboard and mouse input methods, undo/redo, etc.
# Table of Contents

1. System Requirements .................................................................................................................. 1
   1.1 Software .............................................................................................................................. 1
      1.1.1 Minimum ....................................................................................................................... 1
      1.1.2 Recommended .............................................................................................................. 1
   1.2 Hardware ............................................................................................................................ 1
      1.2.1 Minimum ..................................................................................................................... 1
      1.2.2 Recommended ............................................................................................................. 1

2. Features ...................................................................................................................................... 1
   2.1 Edit buffers .......................................................................................................................... 1
      2.1.1 Content ......................................................................................................................... 1
      2.1.2 Representation ............................................................................................................. 1
      2.1.3 Persistent storage ......................................................................................................... 2
   2.2 Clip registers ....................................................................................................................... 2
      2.2.1 Clipboard ...................................................................................................................... 2
      2.2.2 Cut area ......................................................................................................................... 2
   2.3 Markers ................................................................................................................................ 2
      2.3.1 Cursors ......................................................................................................................... 2
      2.3.2 Coordinates ................................................................................................................. 2
      2.3.3 Line numbers ............................................................................................................... 2
   2.4 View ..................................................................................................................................... 2
      2.4.1 Size ............................................................................................................................... 2
      2.4.2 Appearance .................................................................................................................. 3
      2.4.3 Interaction .................................................................................................................... 3
      2.4.4 Navigation .................................................................................................................... 4

2.5 Operations ............................................................................................................................... 4
   2.5.1 Copy ............................................................................................................................... 4
   2.5.2 Cut .................................................................................................................................... 4
   2.5.3 Paste .............................................................................................................................. 4
   2.5.4 Find .................................................................................................................................. 4
   2.5.5 Replace ........................................................................................................................... 5
   2.5.6 Undo ................................................................................................................................ 5
2.5.7 Redo .................................................................................................................. 5
2.6 Syntax awareness .................................................................................................. 5
  2.6.1 Syntax highlighting ......................................................................................... 5
  2.6.2 Bracket matching .......................................................................................... 5
  2.6.3 Scope highlighting .......................................................................................... 6
2.7 Dictionaries ........................................................................................................... 6
  2.7.1 Spell checking ................................................................................................. 6
  2.7.2 Word suggestion ............................................................................................. 6
2.8 Preferences ........................................................................................................... 6
  2.8.1 Editor ............................................................................................................... 6
  2.8.2 Key bindings .................................................................................................... 7
2.9 Help files ............................................................................................................... 7
3. User Interface .......................................................................................................... 7
  3.1 Views .................................................................................................................. 9
    3.1.1 Project manager (A) .................................................................................... 9
    3.1.2 Source code editor (B) ............................................................................... 9
    3.1.3 Terminal (C) ............................................................................................... 11
    3.1.4 Javadoc display (D) ................................................................................... 11
    3.1.5 Code pad (E) .............................................................................................. 11
  3.2 Panels .................................................................................................................. 11
    3.2.1 Menu bar (F) .............................................................................................. 11
    3.2.2 Quick reference buttons (G) ....................................................................... 14
    3.2.3 Program testing features (H) ....................................................................... 14
4. Extension ................................................................................................................ 14
  4.1 Package management ......................................................................................... 14
    4.1.1 Create ......................................................................................................... 15
    4.1.2 Visualize ..................................................................................................... 15
    4.1.3 Modify ......................................................................................................... 15
5. Performance ............................................................................................................. 16
  5.1 Unit Testing ........................................................................................................ 16
  5.2 Conformance testing .......................................................................................... 16
  5.3 Acceptance testing ............................................................................................. 16
References .............................................................................................................................. 17
Direct..................................................................................................................................... 17
Indirect.................................................................................................................................... 17
Appendix .................................................................................................................................. 18
A. Journal ............................................................................................................................... 18
  A.1 Ayushi ............................................................................................................................ 18
  A.2 Ben .................................................................................................................................... 18
  A.3 Kylyn............................................................................................................................... 18
**Table of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>User Interface with Default Views</td>
<td>8</td>
</tr>
<tr>
<td>3.2</td>
<td>User Interface with Alternate Views</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of scope highlighting [8]</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>Project Menu</td>
<td>11</td>
</tr>
<tr>
<td>3.5</td>
<td>Project &gt; New... menu</td>
<td>12</td>
</tr>
<tr>
<td>3.6</td>
<td>Project &gt; Open... menu</td>
<td>12</td>
</tr>
<tr>
<td>3.7</td>
<td>Class menu</td>
<td>12</td>
</tr>
<tr>
<td>3.8</td>
<td>Edit Menu</td>
<td>13</td>
</tr>
<tr>
<td>3.9</td>
<td>Tools Menu</td>
<td>13</td>
</tr>
<tr>
<td>3.10</td>
<td>View menu</td>
<td>14</td>
</tr>
<tr>
<td>3.11</td>
<td>Help menu</td>
<td>14</td>
</tr>
<tr>
<td>4.12</td>
<td>Initial specifications for extension UI</td>
<td>15</td>
</tr>
</tbody>
</table>
1. System Requirements
The BlueJ IDE will be written in Java for beginning Java developers. In order to ensure a successful installation of BlueJ, the intended system must meet certain software and hardware requirements.

1.1 Software

1.1.1 Minimum
- Windows 2000, Mac OS X, Linux/Unix
- Java 5

1.1.2 Recommended
- Windows 7 or 8, Mac OS X, Linux/Unix
- Java 7

1.2 Hardware

1.2.1 Minimum
- 64 MB RAM
- Pentium II processor

1.2.2 Recommended
- 128 MB RAM
- 400 MHz Pentium III processor

2. Features

2.1 Edit buffers

2.1.1 Content

 Encoding
The edit buffer will contain ASCII-encoded text.

 Size
The size of the editor is limited by the amount of memory available.

2.1.2 Representation

 Lines
We are not concerned with how lines are represented internally.

 Edits
Recent edits will be stored in a list. Edits will be defined as the state of a document before a discrete event, such as a defined action, or period of typing delimited by a break in user input of five seconds or more. The list will accommodate at most 100 edit states. Each edit buffer will maintain its own list of edits. The lists will persist until the application is exited.
2.1.3 Persistent storage
The contents of the edit buffer will be stored in a Java file.

2.2 Clip registers

2.2.1 Clipboard

Scope
The clipboard will be the same as that used by the operating system, shared by all applications.

Content
This application will use only the text content on the clipboard.

2.2.2 Cut area
The cut area will capture lines of text in the edit buffer, rather than a rectangular section. A selection that has been cut from the edit buffer will be placed on the clipboard.

2.3 Markers

2.3.1 Cursors

Point
Point will be the current location of the cursor. [1]

Mark
Mark will be the initial location of the cursor which will differ from point only when text is selected. [1]

Region
The part of the buffer between point and mark is called a region and the text within the region is considered to be selected. [1]

2.3.2 Coordinates
The point and mark, each will have Cartesian coordinates in the edit buffer (i.e. row and column).

2.3.3 Line numbers
Each row or line in the edit buffer will be numbered starting with 1 and increasing sequentially by 1.

2.4 View

2.4.1 Size

Rows
The default number of rows visible will be 32. This dimension will be resizable.

Columns
The default number of columns visible will be 103. This dimension will be resizable.
2.4.2 Appearance

Font
The font will be a monospaced serif such as Courier New. The default size will be 12pt, but this will be a user-editable option.

Cursor
The cursor will be displayed as a solid red vertical bar located immediately before the symbol at its position in the edit buffer.

Selection
A selected region of text will be made known by changing the background color of the text. The changed background color is called highlighting.

2.4.3 Interaction

Click
A left-click (pressing and releasing the left button of the mouse) within the editor will relocate the cursor.

Click-and-drag
A left-click hold (pressing and not releasing the left button) will allow selection of text by dragging (moving the mouse while left button pressed) the cursor to highlight text.

Context menu
The context menu is a list of available actions that is dependent on the location from where it is opened [2]. This menu is opened with a right-click (pressing and releasing the right button of the mouse).

Tab
The tab key, when pressed, will insert 4 spaces into the edit buffer at the location of the cursor. Four spaces is a widely followed convention for indenting lines of code.

Enter
With automatic indentation, the enter key, when pressed, will insert a new line and add indentation spaces to match the previous line. The cursor will move to the new line, immediately after the indentation. If the enter key is pressed in the middle of a line of text, the text following the cursor will be moved to the new line still following the cursor. Enter may be pressed in the middle of a statement, splitting it between two lines such that the first so many tokens are on one line while the remaining tokens, ending with the semicolon, appear on the next line. [3]

Key binding
A key binding is a combination of keys that when pressed simultaneously will trigger an action within the editor (e.g. save file, select all text, etc). [4]
2.4.4 Navigation

**Keyboard**
There will be default key bindings that allow the user to navigate the edit buffer using the arrow keys.

**Scrollbars**
A vertical scrollbar will appear when the number of rows in the edit buffer exceeds the number of rows visible. Likewise, a horizontal scrollbar will appear when the number of columns in any line of text in the text buffer exceeds the number of columns visible.

**Preview**
There will be a scaled-down preview of the text displayed on the right side of the edit buffer view which the user can click to jump to anywhere in the edit buffer.

2.5 Operations

2.5.1 Copy
Given a selected region of text, the copy function will create a replica of that text, and store the replica in the clipboard. The point, mark and region will not be modified by this operation. If there already exists some text in the clipboard, the copy function will delete this old text and replace it with the new selection’s replica. If no text is selected, the copy function will not replace any previously-held text.

2.5.2 Cut
Given a selected region of text, the cut function will create a replica of that text, and store the replica in the clipboard. The mark’s location is moved to be that of the point. The region will be eliminated from the edit buffer, but the clipboard will store the replicated region. If there already exists some text in the clipboard, the cut function will delete this old text and replace it with the new selection’s replica. If no text is selected, the cut function will not replace or delete any previously-held text.

2.5.3 Paste
The paste action will make a copy of any text stored on the clipboard and insert that copy into the edit buffer at the location of the cursor. The cursor will be located after the last character of the pasted text once the pasting is complete. If the point and the mark are different, the region between the point and mark is deleted, and replaced with the text from the clipboard. Text on the clipboard is not deleted, and can be re-pasted somewhere else if desired. If no text exists on the clipboard, the paste function does nothing.

2.5.4 Find
The find action will perform a simple exact string pattern match against an input string. It will only search the text file currently selected by the user. This action will produce a list of locations of matched strings, and will display these locations with a highlight. Copy/cut/paste functions will not operate on this highlighted text, unless it is further selected by the user. The find action will handle both case-sensitive and case-insensitive string matching.
2.5.5 Replace
The replace action will remove the text matched with the find action from the edit buffer and in its place insert an input string. Once a string is found and highlighted, the replace action can replace only the particular example of the string currently highlighted, or it can replace all instances of that string.

2.5.6 Undo
After every discrete action (any of the other functions performed, a string of text input, with discrete strings of text being parsed by a five-second delay between typing), the state of the document before that action was performed is added into an edit list. The undo action restores the document to the state it was in before the most recent action was taken, in other words, it removes the top document state from the edit stack, and changes the state of the working document to that of the document as given by the top edit state. This action may be consecutively performed for every item on the list. If the contents of the edit buffer are modified so that an item is added onto the list of edits, any edits that were undone before this change will be lost. When an undo action is performed, before the state of the document is reverted to that of the first undo, its current state is saved to a “redo” stack, which functions exactly like the undo stack, and functions as an anti-undo feature (detailed below). If no edits have been performed, this action does nothing.

2.5.7 Redo
When undo is used, the state of the document, before the undo is applied, is saved to a redo buffer. When redo is selected, it restores the edit that was most recently undone. That is, it replaces the state of the document with the first state in the redo stack. Before the redo action is applied, a copy of the current state of the document is saved to the top of the undo stack. If no edits have been undone, then this action does nothing. The redo action may be performed until the contents of the edit buffer reflect all of the edits in the edit list.

2.6 Syntax awareness
2.6.1 Syntax highlighting
The text will be assigned different colors according to the type of text entity determined by syntactic analysis. These categories are default, standard Java comments (// or /*...*/), non standard stand-out comments (/*#...*/), Javadoc, Java keywords, class-based keywords, method modifiers, other Java keywords (i.e. true, false, this, super), primitives, String literals, labels, invalid (e.g. unclosed String literal). [5]

2.6.2 Bracket matching
The bracket matching feature will help the user realize if he missed out on a closing brace for a function, for example. If the marker is placed immediately following an opening curly brace (or parenthesis), the closing curly brace (or parenthesis) will be highlighted, and vice versa. [6] Performing matching for syntax check is not limited to brackets (curly braces, parentheses, square brackets, angular brackets) but it also extends to other symbols including quotation marks, apostrophes, as well as the character sequences that open and close block comments ("/*", "*/") and Javadoc ("/**", "*/").
2.6.3 Scope highlighting
Different colors of the background regions shall be used to emphasize on the varying levels of scope for better interpretability by the user. It will help maintain a high level of clarity. Nested scopes shall also be supported by scope coloring (e.g. a nested class). Different color codes shall be maintained for each, class level scope, function level scope and scope limiting curly braces. The scope for each function will be highlighted with yellow and that for the entire class will be highlighted with green.

2.7 Dictionaries

2.7.1 Spell checking
BlueJ shall not provide the spell checking feature. The user will have to type in promptly the correct code or when an error is returned during compilation, that is when the error can be rectified. Moreover, owing to syntax highlighting, the chances of making a spelling mistake, with any reserved keywords, are highly reduced as the user can see the visible color difference if the word is spelt correctly and thus avoid misspelt words.

2.7.2 Word suggestion
BlueJ shall not provide for any dictionaries that extend word suggestions helping the user type standard keywords. Although BlueJ will provide with a code outline section that will display the hierarchy of the contents of the file, including classes, methods and variables. The user may refer to this section for obtaining suggestions on the method names and variables of a particular class or a class on the whole.

2.8 Preferences

2.8.1 Editor

Font size
The font size of the text displayed in the editor will be configurable by the user. The only restriction on the size will be that it must be an integer value greater than zero.

Syntax highlighting
Syntax highlighting will be a toggled feature, either on or off. This will be controllable by the user. The colors for highlighting text may be given as hexadecimal values in a properties file.

Line numbers
The display of line numbers may be turned on or off. When turned on, all the lines of code currently present in the edit buffer will be numbered starting from one. These numbers will not hamper with the compiler during the compilation process.

Automatic indentation
The editor shall support automatic indentation within a Java class file. This automatic text indentation feature may be turned on or off. When turned on, the automatic indentation will take place when the user inputs a newline or a tab character. The user will also have the option of automatically indenting an entire Java class file (see section 3.1.2).
Bracket matching
Bracket matching may be turned on or off. When turned on, it checks for any missing brackets within the file currently open in the edit buffer. Bracket checking will be done for all kinds of brackets namely parentheses, curly braces, box brackets and angular brackets. Whenever the marker will be positioned immediately after an opening or closing bracket, its corresponding closing or opening bracket, respectively, shall be highlighted.

Scope highlighting
Scope highlighting may be turned on or off. When turned on, it will help user understand clearly, the scope of all the various curly braces that are open in the current document. Different colors will be used for the purpose of highlighting scopes of different entities like classes and functions. See section 3.1.2 for more details.

2.8.2 Key bindings
Like in any IDE, BlueJ will come with a set of key bindings. But apart from the standard key bindings available, the user will be able to define custom key bindings for file and edit operations, and navigating within the editor.

2.9 Help files
To reduce the size of the program, all help files will be located at bluej.org. The link to these files will be made available through the application. The aforementioned website will help any user download the latest version of BlueJ, refer to its documentation, extensions, resources, version history and visit other related help and information links.

3. User Interface
The BlueJ application will use one window, divided into three main views and three panels [7]. The first of the views will be fixed as the project manager view. The other two views will be occupied by the source code editor and Javadoc display, and the terminal and code pad, respectively. The three panels will contain the menu bar, the quick reference buttons, and the program testing features, respectively (Figures 3.1 and 3.2).
Figure 3.1: User Interface with Default Views

Figure 3.2: User Interface with Alternate Views
3.1 Views

3.1.1 Project manager (A)
The project manager view is where users will create classes and define relationships between classes. They will be able to visualize and organize their projects here without opening a source file. Here the users are encouraged to build the skeleton for their program (i.e. design) before attempting the flesh it out (i.e. implementation). More details about this view will be provided in a separate document that focuses on the project manager specifications.

3.1.2 Source code editor (B)
The source code editor is where users will write and edit the source of the classes belonging to the project that is loaded into BlueJ. The editor will boast scope and syntax highlighting, as well as automatic file save. Additionally it will have tools such as auto-layout, insert method, block comment out, and find/replace. These features have been selected to be included in the source editor because of their usefulness to beginning programmers and their relative ease of use.

Scope highlighting
Scope highlighting will make it easy to see whether brackets are matched with a quick visual scan.

As shown in Figure 3.3, the background color for a particular scope begins with the opening bracket and ends with the closing bracket, which define the scope. Each nesting level will be highlighted with a different color.
Syntax highlighting
The editor will use syntax highlighting to differentiate certain Java keywords (access modifiers, decision structures, etc), primitive types, string literals, boolean literals, and comments.

Automatic file save
Source files will be saved automatically when the editor is closed and before code compilation.

Auto-layout
The auto-layout tool will be used to automatically correct inconsistent indentation in a source file. Each scope in a Java source code file should have a different level of indentation, inner scopes having more indentation. In order for the indentation to be correct, each line within a scope must be at the same level of indentation. The auto-layout tool will use indentation to align each scope at the correct level of indentation. Additionally, the auto-layout tool will correct the size of the indentation itself to four spaces as specified by the Java Code Conventions [9]. If the source file already has correct indentation, a message will be displayed verifying that the indentation was already correct at the time this action was performed. Otherwise, all lines of source code in the file will be corrected.

Insert method
The insert method tool will insert a “template” method into the source code file, starting at the location of the cursor. By template, we refer to a method which has been written in advance and will just be copied into the file. The method will be simple, but still a complete working example. It will have an access modifier, a non-void return type, and at least one parameter. This method will have an accompanying comment block that includes a brief description as well as parameter and return values. The insertion of this method will not cause compilation errors if the method is inserted at an appropriate place (e.g. at the class level).

Block comment out
The block comment out tool will comment out one or more lines of code in a source code file that has been loaded into the editor. When a user wants the compiler to ignore a line of code, but is not ready to delete the line, the user may prepend the line with two backslashes. This is an example of commenting out code. This tool will comment out all lines in a selected region. If the point and mark are on the same line, only that line will be commented out.

Find/replace
The find/replace tool will search a source code file for a given string. When the string is found, the text background of every matched string will be set to blue. The user will be able to navigate the matched strings. The text “in focus,” or currently selected, will have a yellow background color. The user will be able to change which matched string is in focus by navigating forward to the next matched string or backward to the previous matched string. If the user navigates forward from the last matched string, the find/replace tool will continue to the first matched string. If the user navigates backward from the first matched string, the tool will continue to the last matched string.
This tool will also allow the user to replace matched strings with a given replacement string. The replace function will replace the string in focus with the replacement string. There will also be a replace all function that will replace all matched strings with the replacement string.

3.1.3 Terminal (C)
The terminal will behave as a console that has been tasked specifically to run whatever project is loaded into BlueJ. It will be the standard in and out for programs run from BlueJ. The terminal will only accept input for a program running in BlueJ, and only if the program is listening to the terminal for input.

3.1.4 Javadoc display (D)
The Javadoc display will allow the user to see the HTML documentation generated for their class by Javadoc. This view will be read-only, but the user may copy and search the text in this view.

3.1.5 Code pad (E)
The code pad will let the user experiment with Java syntax, accepting both expressions and statements, and producing immediate results. The code pad will accept keyboard input terminated by a newline character. Input text will be black and output text will be green.

3.2 Panels

3.2.1 Menu bar (F)
The menus and menu items will follow the conventions set forth by other IDEs (such as Eclipse), but will be organized to suit beginning programmers. It is for this reason that the application will have 6 menus: Project, Class, Edit, Tools, View, Help.

Project
The Project menu will contain all operations which the user may perform on a project (Figure 3.4).
The Project menu will contain two submenus: New and Open. The New menu will contain all of the operations with which a user will create and add items to a project (Figure 3.5). The Open menu will contain all of the operations with which the user may load an existing project into BlueJ (Figure 3.6).

![Figure 3.5: Project > New... menu](image)

![Figure 3.6: Project > Open... menu](image)

**Class**
The Class menu will contain all of the operations that may be performed on a single source code file (i.e. a Java class). The name of the class in focus will appear next to each item in this menu (Figure 3.7).

![Figure 3.7: Class menu](image)

**Edit**
The Edit menu will contain some convenient tools with which the user may modify the contents of a source code file (Figure 3.8).
Tools
The Tools menu will contain tools that would be useful after a program has been written (Figure 3.9).

View
The View menu will allow the user to toggle the visibility of the different windows and views that are available in BlueJ (Figure 3.10).
Help
The Help menu will contain items that provide the user with information about the installed BlueJ application, as well as links to online resources (Figure 3.11).

3.2.2 Quick reference buttons (G)
Panel G will contain quick reference buttons for operations available in the Project (New Class, Uses Relation, Inherits Relation) and Tools (Compile All) menus. This panel will also include a drop list that will toggle the visibility of the Source Code and Documentation views.

3.2.3 Program testing features (H)
Panel H will contain features useful for program testing, a debugger status graphic and operation status text area [10], as well as a drop list that will toggle the visibility of the Terminal and Code Pad views.

4. Extension

4.1 Package management
A GUI interface will be created which allows for the creation, visualization, and modification of packages.
4.1.1 Create
The GUI will allow for the automatic creation of packages and classes, independently or within other packages. This will be accomplished through either a menu, or through a right click. In either case, the option create new -> package should create a blank package, and create new -> class should create a blank class. If there is no package selected, the new class should appear not connected to any package, but if a package is selected, the class will be created belonging to the selected package. If a package is created when another package is selected, a nested package will be created, or, if no package is selected, a standalone package will be created.

4.1.2 Visualize

![Diagram of class and package relationships]

Figure 4.12: Initial specifications for extension UI

The GUI will allow users to differentiate between classes and packages visually. There will be a blank background, upon which packages can be displayed, along with any classes contained by the package. Classes will appear in smaller font, visually connected to the package by an enclosing dotted line (Figure 4.1). There may also be classes not belonging to a specific package visualized in the GUI, which will simply be classes not contained within a package’s line. A nested package will be represented by a folder icon within the dotted line of the containing package. Double-clicking the package icon will cause the package contents to be displayed in the GUI with a folder icon for going up a level.

4.1.3 Modify
It will be possible to select classes with the cursor, and move a class into a package, which will allow that package to contain the selected class. It will also be possible to make a copy of a class, so that it can be added to one package without removing it from the rest. This will be accomplished by a copy/paste function in the GUI. Adding a class to a package should alter the organization of the project source folders, creating new folders for the new class or package, as well as updating to the source code of the affected classes, both those moved into the package and those classes dependent on it. In essence, it will be as if the class was added, in code, to the package, by the user. These changes will be confirmed by the user before being finalized.
Additionally, a class or package will be able to be deleted, with the appropriate resultant changes to the file structure, and warnings issued.

5. Performance

5.1 Unit Testing
Unit testing in BlueJ will combine BlueJ’s interactive testing functionality with the regression testing of JUnit. Test classes shall be created in BlueJ to test different modules within BlueJ and check for the desired functionality of each [11].

5.2 Conformance testing
A Quality Assurance team will perform conformance testing (compliance testing) to ensure that the developed product (initial version) meets the demands of the BlueJ requirements and specifications documents. Once the set of tests have been completed and the program has been found to comply with all the applicable standards, it will then be released for the Acceptance testing [12].

5.3 Acceptance testing
For BlueJ’s acceptance testing (beta testing), a test version will be available to download over the internet. Once when it meets the selected targeted audience and we have obtained feedback from them about its functioning, the necessary corrections and additions will be made and a better version shall then be released [13]. We will repeat this process for each new release of the BlueJ application.
References

Direct
http://www.emacswiki.org/emacs/AutoIndentation.

Indirect
Appendix

A. Journal

A.1 Ayushi
10/9: Ben, Kylyn and I met and discussed the layout of the document and its expectations. Also started working on the features section. And we divided the responsibilities. (0.75 hours)
10/11: I worked on the Performance section. Gathered some information about unit, conformance and acceptance testing. (2.5 hours)
10/12: Prepared a rough draft for parts of the Features section that I was supposed to work on. The remaining parts are going to be tackled by Ben. (3 hours)
10/13: Finished working on the features section. (1.5 hour)
10/14: Gave some final touches to the Performance section. (1.5 hours)
10/15: Merged journal entries with the document and did some final formatting for submission. (1 hour)

A.2 Ben
10/9 -- Team Planning: Ayushi, Kylyn, and I met to discuss the expectations for this document as well as a possible completion date. (0.75 hours)
10/10 -- Work - looked up specifications documents for other projects, found a good example on Michigan State University’s website, as well as browsing various Spec documents for other open source projects. (2.25 hrs)
10/11 -- Writing - wrote up first drafts of the specifications documents for sections I was responsible for, 2.1 through 2.5. (1.5 hrs)
10/13 -- Writing, reviewed class materials, adjusted first drafts to reflect a strict, non-obfuscated read of requirements - ambiguous language removed (1 hr)
10/14 -- Began work on additional section, Extension, time includes several re-writes and tweaks achieved through discussion with Kylyn (3 hrs)
10/15 -- Finished work on Extension section, reviewed all sections, merged journal entries with this document. (1.5 hrs)

A.3 Kylyn
10/9 -- Team Planning: Ayushi, Ben, and I met to discuss the expectations for this document as well as a possible completion date. We also divided up responsibilities, that is the portions of the document that each of us would be responsible for completing. I am initially responsible for the User Interface, Extension, and Performance sections (0.75 hours)
10/10 -- Writing: Began writing User Interface section. (0.5 hours)
10/11 -- Screenshots: Produced initial screenshots for overview of user interface. (1.5 hours)
10/11 -- Writing: (2 hours)
10/11 -- Team Meeting: Discussed progress (0.25 hours)
10/13 -- Screenshots: Refined initial screenshots and produced some for the menus. (5.5 hours)
10/13 -- Writing: (1.5 hours)
10/14 -- Writing: (5.5 hours)
10/15 -- Writing: (2 hours)
10/15 -- Team Editing: (2 hours)