# Change request log - #je-1

## 1 Team

Team Name: Swathy & Rachit

Rachit Dalal : Change request log tracking

Swathy Hari Prasad : Code inspection, Identification of Concept Location, Impact Analysis, Actualization, Validation and Reverse Engineering

## 2 Change Request

Change Request ID: #je-1

- Modify the Status Bar to show the word offset from the beginning of the file and total number of words in the file.
- Modify the Status Bar Preference option in Global Options menu to add checkboxes of word offset from the beginning of the file and total number of words in the file.

**Github for #je-1 :**

https://github.com/SwathyAravind/cs515-001-s20-Swathy-Rachit-jedit/commit/469fa888a546bdbd422668ffa3346376975a5faa

## 3 Concept Location

| Step # | Description | Rationale |
|---|---|---|
| 1 | jEdit application is run via Eclipse to see it's working and understand the functionality of different options especially where the new feature addition is required. | To get a better understanding of the related code and its functionalities, to identify the concept location of this change request. |
| 2 | The Eclipse Search feature (Ctrl + Shift + R or Open Resource) is used to fetch at least one class where StatusBar Options are used or referred by entering *status* in the search field.<br>The listed classes are:<br>➢ StatusBar.java<br>➢ CheckFileStatus.java<br>➢ HoverSetStatusMouseHandler.java<br>➢ MemoryStatusWidgetFactory.java<br>➢ StatusBarOptionPane.java<br>➢ StatusListener.java<br>➢ StatusWidgetFactory.java | To list the classes where Status is referred to. |
| 3 | From the listed classes, *StatusBar.java* and *StatusBarOptionPane.java* was more related to the feature that we want to add and are inspected further. | To find out where the initial changes are to be made for this change request. |

| | | |
|---|---|---|
| 4 | By inspecting the classes more, we found that StatusBar.java is where the caret position and file buffer length are created and StatusBarOptionPane.java is where the checkboxes in GlobalOptions are given. | Through this we found that in order to add the checkboxes, StatusBarOptionPane.java is to be changed and to add the new word status features, StatusBar.java is to be modified. |
| 5 | StatusBar.java is our identified Concept location where initial modifications to add new features as per the change request are made. | Since we have decided to add the word offset and total words features to Status Bar initially and the checkboxes later on, we identified StatusBar.java as the first place to make the change. |

Time spent (in minutes): 60

## 4 Impact Analysis

| Step # | Description | Rationale |
|---|---|---|
| 1 | StatusBar.java is the first class that we have decided to modify and now we are checking the change propagation. So from StatusBar.java the following classes are referenced.<br>➢ JEdit.java<br>➢ Buffer.java<br>➢ JEditBuffer.java<br>➢ JEditTextArea.java<br>➢ TextArea.java<br>➢ ContentManager.java<br>➢ jedit_en.props<br>➢ View.java | By carefully going through the code, we found out that the given classes are referenced. After analyzing the initially found classes we will reduce the number of impacted classes. |
| 2 | The following classes are impacted from the StatusBar.java class, as the first step.<br>➢ JEditBuffer.java<br>➢ ContentManager.java | In StatusBar.java, the method getlength() is used to get the bufferlength which is used to count the total characters in the file. The method is written in JEditBuffer.java which in turn receives data from the class ContentManager.java. So we know that these two classes are impacted in case of our change request. |
| 3 | Buffer.java and JEdit.java are not modified. | Although Buffer.java is being referenced, there is no change in this class, as it extends from JEditBuffer.java and the modification is made in this class.<br>Methods like “getBooleanProperty()” in JEdit.java are used, to get the show status according to which the status bar options are printed. Other than using the methods of this class, there are modifications needed in this class. |
| 4 | JEditTextArea.java and TextArea.java are not modified. | These classes are used to get the area of selection from the text editor. So they have no modifications. |
| 5 | jedit_en.props is the property file and there are modifications here. This is used for localization in | The jedit_en.props property file is used to display the name of the values shown in the status bar. |

| | | |
|---|---|---|
| | English. SImilarly for other languages, the following files are to be changed.<br>➢ jedit_cs.props<br>➢ jedit_de.props<br>➢ jedit_fr.props<br>➢ jedit_ja.props<br>➢ jedit_ko.props<br>➢ jedit_ru.props<br>➢ jedit_zh.props | When we hover around the values shown in the status bar, it is shown as “line, column[-virtual] (offset/length)”. To add the wordOffset and length, this file is changed. |
| 6 | View.java has no modifications. | View.java is used to get the overall view of the JEdit text editor without concentrating more into the status bar options. So there are no modifications required. |
| 7 | The second portion of this change request is to add checkboxes. For that the impacted classes are:<br>➢ StatusBarOptionPane.java<br>➢ AbstractOptionPane.java | From analysis, we found that to add the checkboxes, the given classes are used. |
| 8 | StatusBarOptionPane.java is impacted and AbstractOptionPane.java is not impacted. | Checkboxes are created in StatusBarOptionPane.java using java swing components. So it requires modifications to add the toggle checkboxes.<br>StatusBarOptionPane extends AbstractOptionPane and there are no changes required to be made in AbstractOptionPane |
| 9 | EditPane.java and View.java uses the method updateCaretStatus() that needs modification in StatusBar.java.<br>But they are also not impacted. | These classes are simply using updateCaretStatus() method for display. So does not need modification. |

Time spent (in minutes): 150

## 5 Prefactoring (optional)- Not Done

| Step # | Description | Rationale |
|---|---|---|
| 1 | | |

Time spent (in minutes): 0

## 6 Actualization

| Step # | Description | Rationale |
|---|---|---|
| 1 | We started by modifying StatusBar.java class, as this is where this class directly affects the status bar options. Inside updateCaretStatus() we first added more else if statements and hardcoded the values to see if the appended values are getting reflected in the status bar. | This is done so that we can know exactly where the changes are to be made and how it gets affected in the status bar. This is like a trial and error method, done prior to actual coding. |

| | | |
|---|---|---|
| 2 | We also changed the value of attribute “caretTestStr” to "9999,999-999 (99999999/99999999) (99999999/99999999)" instead of "9999,999-999 (99999999/99999999)” to display the word offset and total length in correct format. | This is done to display the word offset and total length in correct format in the status bar.. |
| 3 | Now we ran the application and got the details printed with hard-coded values. | This verified that whatever we analyzed so far is correct. |
| 4 | Once we got the hardcoded values displayed in the status bar, we modified ContentManager.java class to add a method “getWordStatus(int)” that will receive the caret position of the selected character and returns the word offset of that particular caret, according to what is passed on to it from JEditBuffer,java which in turn gets the value from StatusBar.java<br><br>This same method will return the total number of words also, as then we will pass the buffer length that is already in use. and the caret position will be at the end of the file. | From impact analysis, we got an idea about the control flow and we know that the major change of adding a new method is to be done in ContentManager.java. |
| 5 | The method added in ContentManager,java is to be called in JEditBuffer.java by creating a similar method “getWordStatus(int)”. This method will return the word position according to the caret position passed on to it from StatusBar.java. | Again from impact analysis, we know that the changes done in ContentManager is propagated to StatusBar.java through JEditBuffer.java. For this we call the new ContenManager method in JEditBuffer by creating a new method in JEditBuffer, but of the same name getWordStatus(int). |
| 6 | Now that new methods are created in ContentManager and JEditBuffer, we used them to create and initialize variables “wordCount” and “wordPosition”. We replaced the hard-coded values with these variables in the method updateCaretStatus. | Since we know where to insert the values, we removed the hard-coded values and replaced them variables. To these variables, the data is stored using the new method we created. |
| 7 | We ran the application and the values got printed in the correct format. But when we hovered around those values, it showed only “line, column[-virtual] (offset/length)”. So this time we changed the localization files that we have found during the impact analysis. We modified “jedit_en.props” which is the localization used for English language. Other property files mentioned in impact analysis (Step 5) need changes, but since they are not in English language, we skipped those modifications. | We edited this property file to add the name as “line, column[-virtual] (offset/length) (wordOffset/length)”. So now if we keep the mouse pointer on top of the values shown in status bar, it shows this modified name. |

| | | |
|---|---|---|
| | Now running the application gave us a perfect result as per the requirement. | |
| 8 | For the second part of the change request, to add toggle checkboxes, we modified the StatusBarOptionPane.java class.<br>● First we created the checkbox private variables “showWordBufferLength'' and “showWordOffset” and then instantiated using JCheckBox.<br>● Also, gave a description to be shown for these checkboxes.<br>● Then we added them to optionsPanel using addComponent()<br>● Then we set the property if the checkbox is selected. Only if the checkboxes are selected, the data will be shown in the status bar. This is done through jEdit.setBooleanProperty() in StatusBarOptionPane.java class and jEdit.getBooleanProperty() in StatusBar.java class. | This is done to add checkboxes in Status Bar Preference option in Global Options menu for word offset from the beginning of the file and total number of words in the file. |

Time spent (in minutes): 90

## 7 Postfactoring

| Step # | Description | Rationale |
|---|---|---|
| 1 | Initially in ContentManager.java and JEditBuffer.java, we wrote two new methods to calculate word offset and total number of words. The two methods had duplicated code with very small changes required for word offset and total number of words. | Followed the same format that the code originally had by creating two different methods for each functionality. |
| 2 | Once we got the code and functionalities working, we tried combining both the methods together by introducing a method variable to be passed on to the method.<br>This is how we reached the latest code with one single method “getWordStatus(int)” and we get different output based on the input that we give. | Keeping the functionality, we got a better and much efficient code just by introducing a variable. in the method declaration. |

Time spent (in minutes): 30

## 8 Validation

Sample text used: Given at the bottom of this document

| Step # | Description | Rationale |
|---|---|---|

| | | |
|---|---|---|
| 1 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Click on the last character (character position = 6032)<br>● Expected Output : last values of the string in Status bar will be (936/936).<br>● Actual Output : last values of the string are (936/936).<br>● Test Result : Passed | Checks if the status bar shows full word count and the last word position for the last character in the file. |
| 2 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Click on the first character of last word (character position = 6026).<br>● Expected Output : last values of the string in Status bar will be (936/936) as it is still the last word referred.<br>● Actual Output : last values of the string are (936/936).<br>● Test Result : Passed | Checks if the status bar shows full word count and the last word position for the first character of the last word in the file. |
| 3 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Add 2 or 3 spaces. Character offset and character count changes.<br>● Expected Output : word count should remain as 936 and last value of the string should be (*/936).<br>● Actual Output : word count is 936 and last value is (*/936)<br>● Test Result : Passed | Checks if the status bar full word count does not change when spaces are added. The spaces are considered as characters but not as words. |
| 4 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Add 2 or 3 lines. Character offset and character count changes.<br>● Expected Output : word count should remain as 936 and last value of the string should be (*/936).<br>● Actual Output : word count is 936 and last value is (*/936)<br>● Test Result : Passed | Checks if the status bar full word count does not change when lines are added. The lines are considered as characters but not as words. |

| | | |
|---|---|---|
| 5 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Click before the first letter of the first word.(character position = 0)<br>● Expected Output : word position should be 0 and the last values should be (0/936)<br>● Actual Output : word position is 0 and the last values are (0/936)<br>● Test Result : Passed | Checks if the status bar shows the word offset of 0th letter (cursor placed before first letter) as 0. |
| 6 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Do not paste anything or keep the text editor empty.<br>● Expected Output : word position and word count should be 0 since character count and character position is 0. The last values of the string look like (0,0).<br>● Actual Output : word position and word count are 0. The status bar string looks like 1,1 (0,0)(0,0)<br>● Test Result : Passed | Checks if the status bar shows 0 as the word offset and total word count when the file is empty. |
| 7 | Open JEdit Application. Make sure that the Status Bar Preference option in Global Options menu has both the checkboxes for word count and total words checked.<br>● Inputs : Add or paste a text with 936 words. Select the first two lines with the cursor at the end of the second line.<br>● Expected Output : word position should be 32, as there are 32 words in the first 2 lines and total word count should be 936. The last values of status bar string should look like (32/936)<br>● Actual Output : word position is 32 and the last values of the string look like (32/936).<br>● Test Result : Passed | Checks if the status bar shows the correct word offset and word count when a part of the text is selected. |
| 8 | Open JEdit Application. Add or paste a text with 936 words and 6033 characters. Go to Utilities->Global Options->jEdit->Status Bar.<br>● Expected Output : 2 additional checkboxes of “Show word offset from start of file” and “Show number of words in the file” should be present.<br>● Actual Output : Checkboxes, “Show word offset from start of file” and “Show number of words in the file” are present. | Checks if the status bar shows both word offset and word count when the both the checkboxes are checked in the preference option. |

| | | |
|---|---|---|
| | • Test Result : Passed | |
| 9 | Open JEdit Application. Add or paste a text with 936 words and 6033 characters. Go to Utilities->Global Options->jEdit->Status Bar.<br>• Input : Uncheck both the checkboxes and click on Apply.<br>• Expected Output : word position and word count should not be shown in the status bar.<br>• Actual Output : word position and word count are not shown in the status bar.<br>• Test Result : Passed | Checks if the status bar does not both word offset and word count when the both the checkboxes are unchecked in the preference option. |
| 10 | Open JEdit Application. Add or paste a text with 936 words and 6033 characters. Go to Utilities->Global Options->jEdit->Status Bar.<br>• Input : Uncheck “Show word offset from start of file” and check “Show number of words in the file”<br>• Expected Output : word position should not be shown and word count should be shown in the status bar as (936).<br>• Actual Output : word position is not shown and word count is shown as (936).<br>• Test Result : Passed | Checks if the status bar does not show word offset and shows word count when the checkbox “Show word offset from start of file” is unchecked and “Show number of words in the file” is checked in the preference option. |
| 11 | Open JEdit Application. Add or paste a text with 936 words and 6033 characters. Go to Utilities->Global Options->jEdit->Status Bar.<br>• Input : Check “Show word offset from start of file” and uncheck “Show number of words in the file”. Keep the cursor on the 7th word in the second line.<br>• Expected Output : word position should be shown as (7) and word count should not be shown in the status bar.<br>• Actual Output : word position is shown as (7) and word count is not shown..<br>• Test Result : Passed | Checks if the status shows word offset and does not word count when the checkbox “Show word offset from start of file” is checked and “Show number of words in the file” is unchecked in the preference option. |

Time spent (in minutes): 60

## 9 Timing

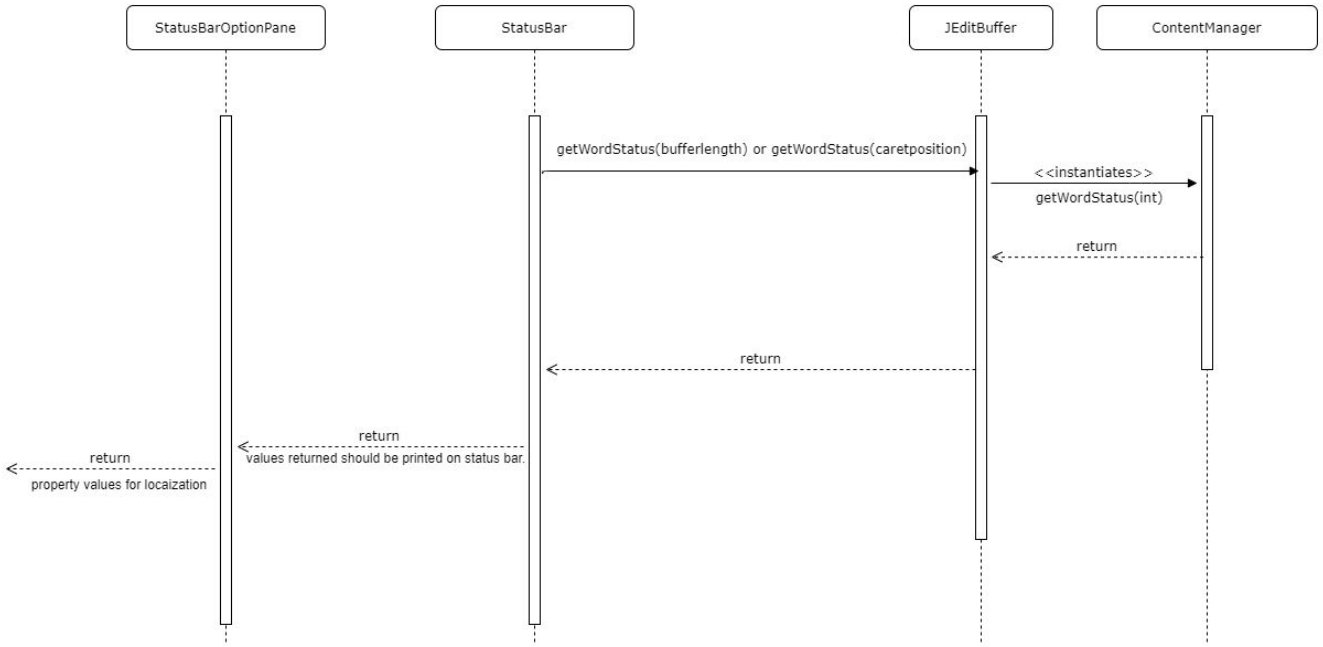
Summarize the time spent on each phase.

| Phase Name | Time (in minutes) |
|---|---|
| Concept location | 60 |
| Impact Analysis | 150 |
| Prefactoring | |

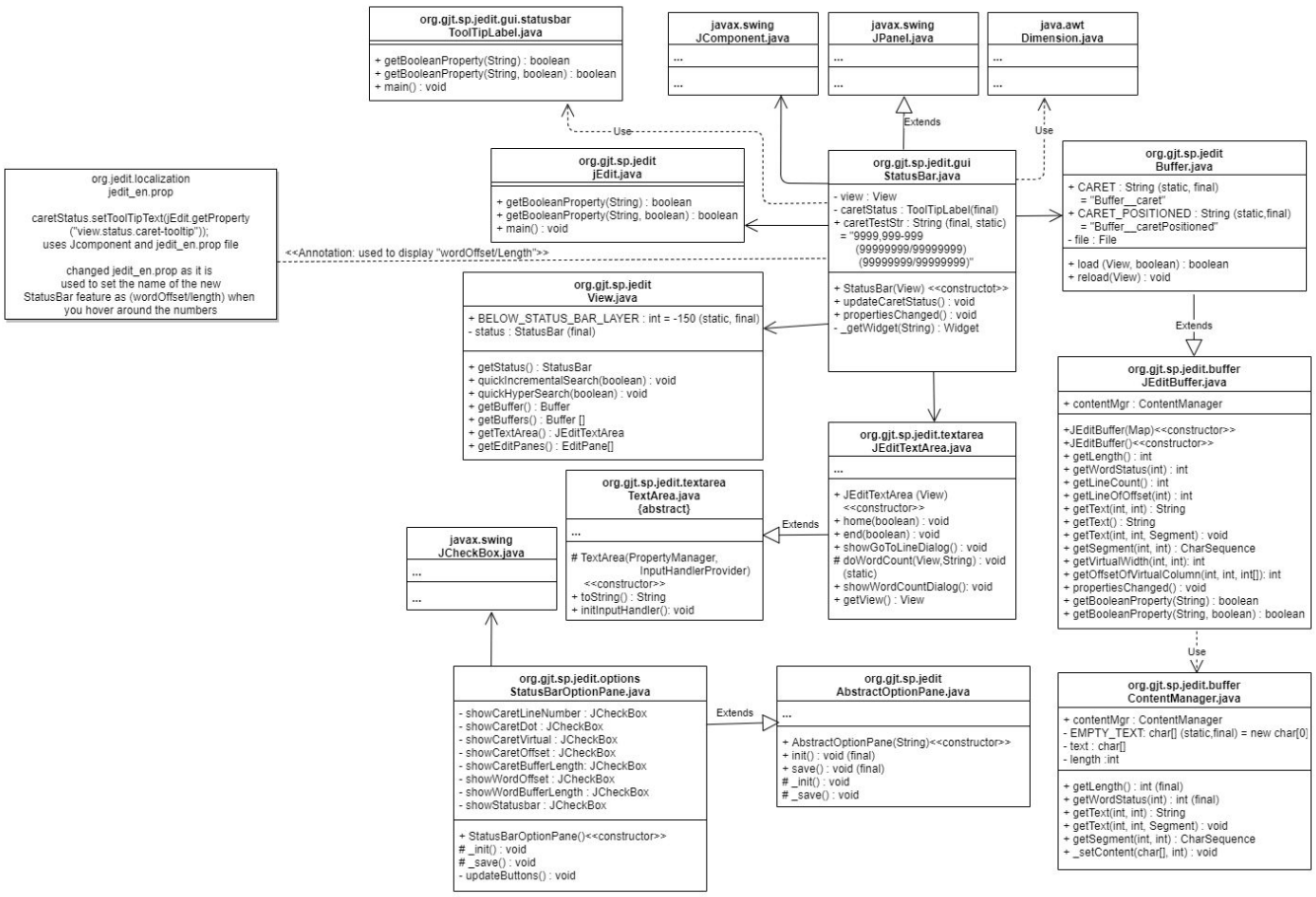
| | |
|---|---|
| Actualization | 90 |
| Postfactoring | 30 |
| Verification | 60 |
| Total | 390 |

## 10 Reverse engineering

UML Sequence Diagram




UML Class Diagram




## 11 Conclusions

Finding Concept location for this feature was comparatively easy by using the Eclipse Search feature. We have not used any other tools for identifying concept location or impact analysis. Impact Analysis took a little more time. Along with manually checking the code we also used a File Search to check if the desired attributes and methods are used elsewhere. We made it very certain that no changes are missed out and everything is working even after the change.

The following classes, methods and attributes where modified:

- org.gjt.sp.jedit.gui.StatusBar.java
  - ➢ Added attributes wordCount and wordPosition that calls getWordStatus(int)
  - ➢ Modified method updateCaretStatus()
- org.gjt.sp.jedit.buffer.JEditBuffer.java
  - ➢ Added method getWordStatus(int) that returns an int value.
- org.gjt.sp.jedit.buffer.ContentManager
  - ➢ Added method getWordStatus(int) that returns an int value.
- org.jedit.localization.jedit_en.props

- ➢ changed view.status.caret-tooltip to “line, column[-virtual] (offset/length) (wordOffset/length)”
- ➢ Added options.status.word.offset and options.status.word.bufferlength

- org.gjt.sp.jedit.options.StatusBarOptionPane.java
  - ➢ Added attributes showWordOffset and showWordBufferLength using JCheckBox and initialized appropriately.

**Sample Text used for Validation:**

Features

jEdit includes syntax highlighting that provides native support for over 200 file formats. Support for additional formats can be added manually using XML files. It supports UTF-8 and many other encodings.

It has extensive code folding and text folding capabilities as well as text wrapping that takes indents into account.

The application is highly customizable and can be extended with macros written in BeanShell, Jython, JavaScript and some other scripting languages.

Plug-ins

There are over 150 available jEdit plug-ins for many different application areas.

Plug-ins are used to customize the application for individual use and can make it into an advanced XML/HTML editor, or an integrated development environment (IDE), with compiler, code completion,

context-sensitive help, debugging, visual differentiation and language-specific tools.

The plug-ins are downloaded via an integrated plug-in manager which finds and installs them along with any dependencies. The plugin manager will track new versions and can download associated updates automatically.[5]

Some available plug-ins include:

Spell checker using Aspell

Syntax and style checkers for various languages[6]

Text auto-complete

Accents plugin that converts character abbreviations for accented characters as they are typed.

XML plugin that is used for editing XML, HTML, JavaScript and CSS files. In the case of XML, the plug-in does validation. For XML, HTML and CSS, it uses auto-completion popups for elements, attributes and entities.[7]

Reception

In general jEdit has received positive reviews from developers.

Rob Griffiths wrote in April 2002 for MAC OS X HINTS saying he was "very impressed" and naming it "pick of the week". He cited its file memory upon reopening, its ability to notice if an open file was

changed on disk by another program, syntax coloring, including that users can create their own colour schemes, split windows feature, show line number feature, convertible tabs to soft-tabs and view sidebars.

He also praised its customization possibilities using the extensive preferences panel and the "on the fly" search engine, which searches while typing. Griffiths noted that the application has a few drawbacks,

such as that it is "a bit slow at scrolling a line at a time" and that because it is a Java application it doesn't have the full Aqua interface.[8]

Also reviewing the application in April 2002, Daniel Steinberg writing for O'Reilly Media said "The strength of jEdit for Java developers comes from the plug-ins contributed by the community...For

the most part, there's nothing here that couldn't be done with BBEdit or even with Emacs or vi. jEdit packages the capabilities much more nicely and makes it easy to call often-used functionality using the plug-ins.

Where I saw NetBeans as overkill, others may see jEdit as underkill for an IDE or overkill for a text editor. I find it Mac friendly and easy to use. I don't expect too much from it, so I tend to be pleased with what I get."[9]

Scott Beatty reviewing jEdit on SitePoint in 2005 particularly noted the application's folding feature along with its search and replace and PHP syntax highlighting capabilities.

He recommended the use of the PHPParser plug-in. PHPParser is a sidebar that checks for PHP syntax errors whenever a PHP code file is loaded or saved.

He noted that downloading jEdit is simple, but that getting and installing the plug-ins to customize it for individual use can be a complex process:

"Beware that a full setup requires a series of downloads, and that this process can take time."[10]

Writing in December 2011, reviewer Rares Aioanei praised jEdit's versatility, stating "jEdit's design allows you to use it as a simple editor, but also use it as an IDE and expand its functionality via plugins

so that it becomes exactly what you want it to be for the task or language at hand." but also adding that "jEdit is not, however, an IDE with everything but the Christmas tree, like Eclipse or Microsoft Visual Studio.

Rather, it's a compact application for editing code, providing practical tools along with basic IDE features."[11]

DESCRIPTION

Artificial intelligence (AI) has been fluidly defined, but, broadly, definitions have focused on either replicating human-like thought/behavior or creating rational thought/behavior. Historically, the goal of AI was to create a general intelligence (GI). when it became clear GI was out of reach, AI split into numerous subfields like computer vision and natural language processing, to name a few. Within the last decade, each of these subfields has achieved breakthrough results, facilitating AI applications like autonomous vehicles and home assistants. These successes have caused some to speculate the replication of GI in a computer is just around the corner. However, the reality of the situation is quite the opposite: these applications, while groundbreaking, are not indicative progress toward the original goals of intelligent thought or action. In this course, we will discuss the limits and potentials of various approaches to modern AI applications, as well as opportunities to overcome these limitations. In this vein, this course will be a tour of both classic ideas and state-of-the-art methods in AI.

LEARNING OUTCOMES

Upon successful completion of the course, students will be able to:

Understand the philosophical underpinnings of the field.

Discuss which problems that are solvable with today's AI algorithms and others that require novel solutions.

Deploy general search algorithms that can be applied to a wide variety of tasks.

Formulate decision making processes that can be used for planning and classification purposes.

Build intelligent agents that perform simple tasks in an autonomous fashion.

Learn task-specific models from large collections of labeled training data samples using algorithms that are optimized using numeric solvers.

Analyze the successes and limitations of an AI system, with an emphasis on real-world applications and ethics.