Keeping Functions Inline:
An Eclipse Plug-in
Individual Requirements Analysis Document

https://github.com/mfraser4/comp195-spr19

Mark Fraser
m_fraser3@u.pacific.edu

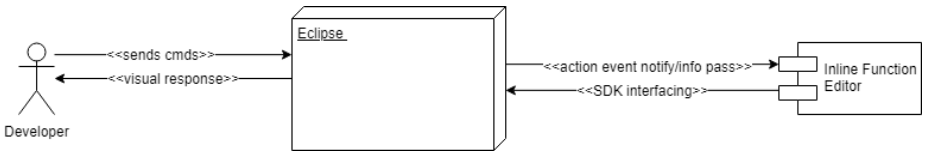
February 20, 2019

Contents

Chapter 1

System Overview

1.1 Basic Architecture



This section is still in flux. Initial analysis suggests that the plug-in will be an Eclipse workbench tool that interfaces with whatever current workspace environment the developer is in and the Eclipse UI. Both of these can be represented as objects in code, and the project itself will be done in Java, an inherently Object-Oriented Programming language. Thus, UML is an appropriate method to describe our project architecture once the software solution is better understood. Eclipse plug-in development/SDK usage documentation is likely to be the single most important document required to derive an elegant solution for an inline/pseudo-inline function editor (*Eclipse Foundation*, n.d.). Above is a basic architectural layout of our proposed plug-in. Note that the plug-in is completely removed from the actor and only interfaces with Eclipse. The user can access the plug-in's functionality, but only through the intermediary that is the foundational framework of Eclipse.

Referencing the inter-woven nature of the use cases, the plug-in should support state machine functionality if some of the implied functionality of the use cases do not follow normal conventions. This is proposed given the number of different paths with which use cases can be started. Some use cases are not executed in isolation, and various efficiency-improvement functionality and error-handling should be implemented to handle these likely edge cases if possible within the time frame of this project.

1.2 Motivation

Software engineers' and programmers' services alike are in constant demand in today's technologically centered society. Because there is insufficient supply to meet demand, software development personnel command higher salaries. Thus, from a company's perspective, software engineers and programmers should be supported as much as possible to maximize their efficiency. From the software engineer's and programmer's perspective, developing software should be as seamless a process as possible to facilitate implementing their code logic. Many programmers develop in Integrated Development Environments (IDEs) to leverage shortcuts and plug-ins to streamline their workflow. Eclipse is one of the most popular open-source IDEs with many plug-ins, but several key features are missing that have been implemented in other IDEs and are desired by developers. Our goal is to implement one of these features: inline function editing.

1.3 Purpose

The proposed plug-in will allow programmers to retain the context of the function they are currently developing while simultaneously viewing a called function's contents. This will be done by displaying the called function's contents inline or pseudo inline with the calling function's contents. Resultingly, fewer button presses, fewer mouse clicks, and less reorientation will be required for developers to work on two coupled functions.

Chapter 2

Current State of the Art

2.1 Eclipse

Currently, Eclipse does not support inline function editing. In order to view and edit a called function's contents (assumed to be in another file for this use case), the following steps are executed using keyboard shortcuts:

1. Position cursor over called function
2. Press <F3>
3. Current development pane shifts to called function's file
4. Cursor is placed at function declaration
5. Developer makes changes to called function
6. <Ctrl+S> to save contents
7. <Ctrl+W> to close called function's editor pane
8. User is returned to original function pane

Using mouse clicks, the following steps are required to perform the same action:

1. Position cursor over called function
2. Open *Navigate* menu
3. Select *Open Declaration*
4. Cursor is placed at function declaration
5. Developer makes changes to called function
6. Open *File* menu

7. Select *Save* option

8. Click the "x" on the called function's editor pane

9. User is returned to original function pane

This requires 8 and 9 steps, respectively. Also, both use cases require the user to reorient themselves to a new editor pane. If the developer needs to go back and forth between the files, this adds 3 steps for each iteration between:

1. Switching to other function editor pane

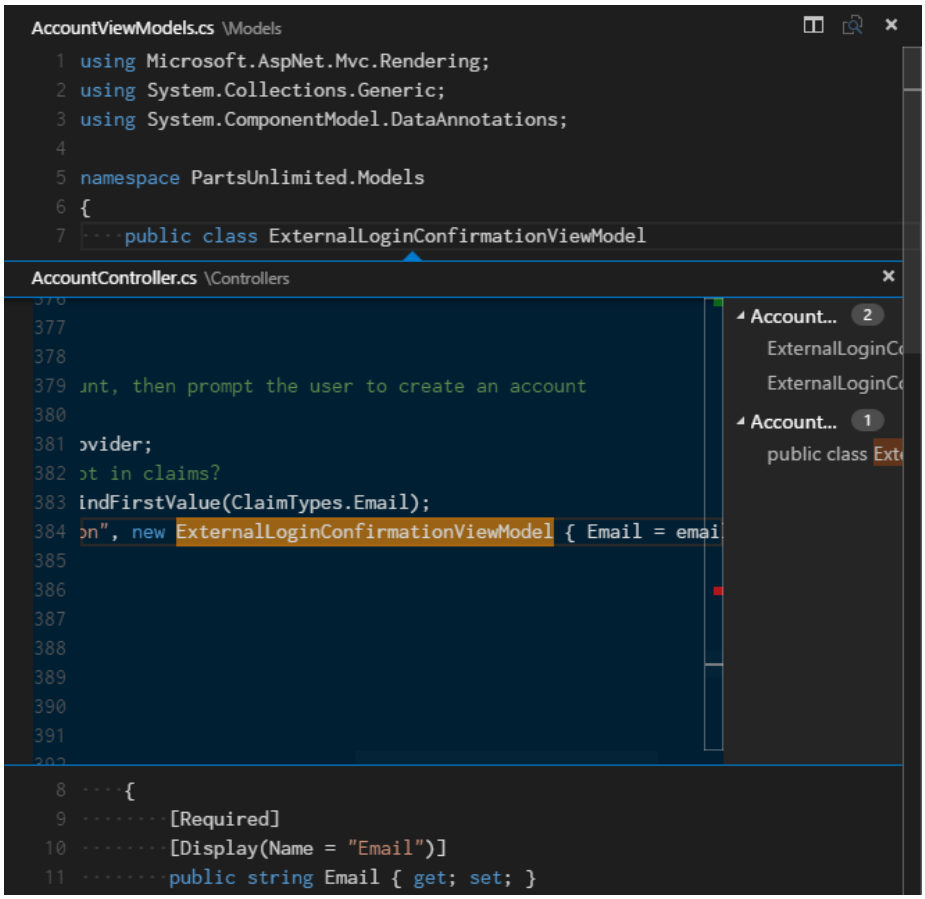
2. Making changes/reviewing code

3. Switching back to original function editor pane

This requires much overhead to view and/or edit two functions simultaneously. Some might argue that one can simply place the editor panes side-by-side, but this use case requires one to clutter the screen with dense text, which is not desirable for a UI. This will still diminish efficiency, as more text to process requires more of the user's energy to parse through. As will be made clear in the following sections of other IDE's current implementations, inline function editing is the best use case of the three explored in this section.

2.2 Visual Studio 2017

Below is a snapshot from an instance of the inline function editor's usage:



Visual Studio 2017 does support inline function editing through the following steps (*Downloads and Tools for Windows 10*, n.d.):

1. Position cursor over called function
2. Press <Alt+F12>
3. Called function's file is opened to function's declaration within text below
4. Cursor is placed at function declaration
5. Developer makes changes to called function
6. <Ctrl+S> to save contents
7. User clicks the "x" to close the called function editor box

8. The inline function editor box is closed, returning the pane to its original state

One can observe benefits not necessarily clear, as the number of steps remains the same. The function is opened within the pane. Therefore, the only change from the user's perspective is a box of text opening beneath the called function, rather than an entirely new pane opening. Also note that the original function can still be reviewed without any physical transitions. Mental and physical overhead are not required to completely switch contexts between files. Instead, the steps to switch between function editing is:

1. Click outside inline function editor
2. Making changes
3. Click inside inline function editor

Clearly, this is a more seamless transition between the two functions than Eclipse currently possesses.

There is one mark against Visual Studio. Whether it be because it is poorly documented or simply unavailable, it appears a transition from the keyboard to the mouse and back is required to close the inline editor pane. As a Vim and general non-IDE user looking to find a similar, yet more facilitated experience, this is a notable point of inefficiency and distraction. In the proposed plug-in's implementation, the insertion of a keyboard hotkey to close the inline pane should be implemented if possible.

Chapter 3

Project Stakeholders

3.1 Product Owner

The owner of the plug-in. This stakeholder is responsible for delivering a product that meets the standards of Eclipse plug-ins and can be added to the plug-in store. This stakeholder is also responsible for maintaining the integrity of the code base and managing possible pull requests from outside collaborators, as this project is open-source.

3.2 Client

Eclipse can be considered the client if it adopts this plug-in as a part of its plug-in store. Eclipse is an open-source IDE, so it does not have any business stake except that it might receive more downloads if it possesses inline function editing.

The University of the Pacific can also be considered a stakeholder in this project, as the success of this project can potentially be profitable in building the school's reputation in computer science by increasing U.o.P.'s presence in the computer science field. The resulting product will also be displayed on Senior Project Day, so the SOECS will benefit from this product's success when displayed to the rest of the school.

3.3 Users

There is one general user of this plug-in, and that is users of Eclipse. Eclipse users might be drawn to this IDE because of its low overhead costs compared to others (e.g. Eclipse requires only 360 MB storage and 3 MB of RAM to run, whereas Visual Studio requires 2.3 GB of storage and 250 MB of RAM). The proposed plug-in contained herein provides similar functionality to comparable IDEs with substantially less overhead.

There are no levels of privileges, as Eclipse is an open-source IDE, and there are no different levels of functionality to imply different clients (i.e. − paid versus unpaid). Users of this plug-in will use it with the intention of increasing their development efficiency through better code navigation than what Eclipse currently offers. Without any data to support this other than anecdotal evidence, keyboard-centric users might be more predisposed to utilize this functionality (e.g. Those experienced and most comfortable with Vi, Emacs, and other similar terminal editors).

Chapter 4

Non-Functional Requirements

4.1 Constant Uptime

Users should be able to access this functionality at any time that Eclipse is open. The plug-in must inherently be stable to achieve this requirement.

4.2 Efficiency and Minimalism

The plug-in's usage should match its purpose, which is to improve efficiency of code viewing and editing. Thus, the interfacing mechanisms involved should be implemented with minimal effort on the developer's side, both mentally (memorizing hotkeys, switching editor panes, et ceters) and physically (key strokes, keyboard-to-mouse movements, and vice/versa).

Developers are most efficient when utilizing a consistent method of human-computer interaction. Of the available tools, the keyboard is the most efficient interface, as most all developer tools have been created with keyboard hotkeys, and manually generated code is created via the keyboard. Also, developers are very likely to spend time interfacing through CLI with other devices, so consistency is key. This is opposed to switching between using the keyboard and mouse, or keyboard and touchscreen, or all of the above. Thus, a plug-in dedicated to efficiency should therefore provide some hot key functionality on the keyboard to best fulfill its goal.

4.3 Multiple Paths of Execution

For developers who do not wish to bind this plug-in's functionality to the main use case(s), they must still be able to interface with the plug-in through the

Eclipse menus. Therefore, menu options and/or other usage cases must be implemented to guarantee interfacing functionality.

4.4 Efficient Content Display

The plug-in's purpose is to display text inline or pseudo inline with the current file to increase developer efficiency. Thus, the contents of the function to be displayed must be shown in an efficient manner such that users can easily switch between the called function's contents and the calling function's contents.

4.5 Failure Management

Some functions might not exist or be findable. Proper error-handling and user informing should be established to effectively communicate this to the user.

4.6 Portability

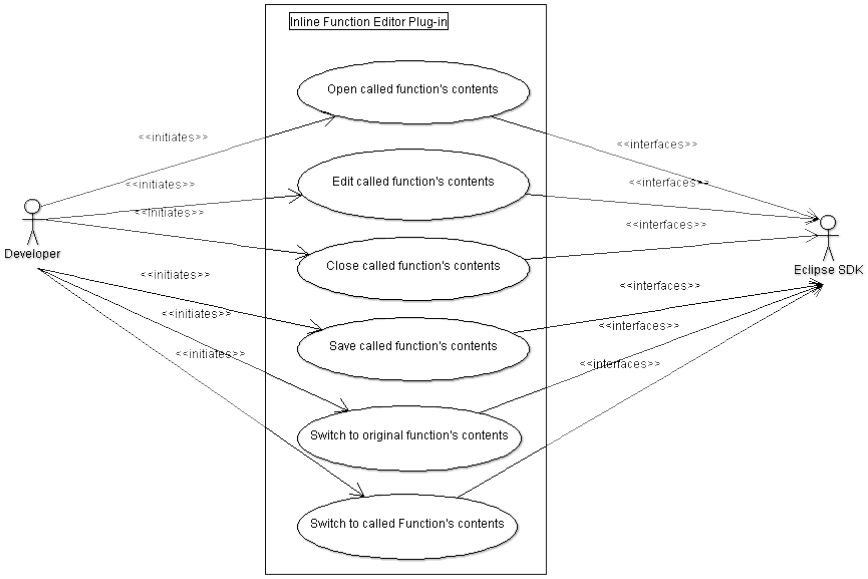
Referring to portability between Eclipse versions. The plug-in should be architected to maintain functionality and require minimal changes when ported to future versions of Eclipse.

Chapter 5

Functional Requirements

5.1 Use Case Diagram

The use cases described in the diagram on the following page can be argued to be part of one overall larger use case, which is editing called functions inline with the original function. However, given the high focus on efficiency, a finer granularity is required to identify how to best execute this overall use case with the minimum amount of key strokes/button presses and the most intuitive interfacing methods. Also, do misinterpret minimalist use case descriptions/steps to result from a desire to minimalize functionality. Rather, these steps are born from a desire to simplify and minimalize the plug-in's presence. It's goal is to facilitate development, not come under the spotlight of users. This implies that more work will have to be performed on the plug-in's backend functionality to best facilitate these use cases.



5.2 Casual Use Case Descriptions

UC1: Open called function's contents

The user, upon positioning his cursor over the called function, can open either the *Navigate* menu and select a "Peek Function" option, or press some keyboard hotkey (preferably one button). An editor pane contained within the original pane should open with the called function's file opened to the declaration of the called function. If the called function is declared within the same file, a duplicate of the file should be opened, but synchronicity should be maintained between the two versions of the file to avoid race conditions. Perhaps for version 0.0, this functionality should be prevented, as this is the most complicated use case, or at least documented to be unsupported. A warning label could be shown to notify the user of possible error in editing the same content between two panes. If the function is recursive and calls itself, it might be most prudent to disallow this functionality as well for version 0.0. Also, if the user attempts to press the same hotkey on a called function within a called function's editor, the plug-in should go to **UC3**. Supporting nested inline editors opens many complicated use cases and alternate states. Upon the completion of this use case, **UC1**, **UC5** should follow naturally.

UC2: Edit called function's contents

An implied use case for this plug-in. This use case should be satisfied following standard Eclipse document editing to mesh with the general Eclipse workflow.

For example, clicking, keyboard navigation, typing, et cetera, should all be supported. There is a possible race condition of this file being opened in a separate editor pane and the user making unsaved changes to this file first. If possible, a file listener should know when changes are made and be reflected in the inline plane, if possible, or make the user refresh the file and reapply his changes.

UC3: Close called function's contents

Once the user has completed whatever his purpose for opening the called function's declaration, he should be able to close the inline editor pane. This should be supported at least two ways: keyboard and an "x" button. If the file has been modified but not saved, **UC4** should be consulted first. This use case and, if applicable, **UC4** should be consulted if the original editor pane is closed. **UC4** should be executed upon the completion of this use case.

UC4: Switch to original function's contents

An implied use case for this plug-in. Leveraging the Eclipse SDK UI components should automatically allow for this functionality. This use case is dependent on the user being in the called function's editor pane. This use case should also be executed upon the completion of **UC3**.

UC5: Switch to called function's contents

An implied use case for this plug-in. Leveraging the Eclipse SDK UI components should automatically allow for this functionality. This use case is dependent on the user being in the original function's editor pane. This use case should also be executed upon the completion of **UC1**. If it is executed as a result of **UC1**, the cursor should preferably be placed at the function declaration line.

5.3 Fully Dressed Use Case Descriptions

UC1: Open called function's contents

Preconditions:

1. Cursor positioned over word in open file
2. No other inline pane is open

Use Case Scenario:

1. User makes call to open inline function editor

   User presses associated keyboard hotkey

   User goes to *Navigate* menu option and clicks on associated sub-option

2. If valid function located in separate file, go to step 3, else 6
3. Eclipse editor pane opens
4. **UC5**
5. Go to end
6. Display error message corresponding to following conditions:

   If not a valid function (e.g. undeclared, variable, et cetera), display to user that function could not be found

   If function is declared within file, display to user that functions declared within document cannot be edited inline (perhaps suggest they press <F3> instead to jump to declaration)

   If the called function is the same as the original function, display to user that recursive function calls cannot be edited inline

   Display to user an unknown error has occurred
7. End

Postconditions:

1. None

UC2: Edit called function's contents

Preconditions:

1. **UC1**
2. Cursor is located within inline editor pane

Use Case Scenario:

1. User makes changes in accordance with standard Eclipse functionality to navigate and edit text
2. End

Postconditions:

1. None

UC3: Close called function's contents

Preconditions:

1. **UC1**

Use Case Scenario:

1. User presses the hotkey to open the called function
2. If file contents have been changed, prompt user to ask if they wish to save new version of file.
3. If user answers "yes," go to step 4, else go to step 5
4. Save inline editor contents to file on disk
5. If cursor is located within inline editor pane, **UC4**
6. The inline editor pane closes
7. End

Postconditions:

1. None

UC4: Switch to original function's contents

Preconditions:

1. **UC1**

Use Case Scenario:

1. If resulting from **UC3**, jump to step 3
2. User uses one of the following methods to switch to original function's contents:

   Standard mouse click to switch to original editor pane

   An intuitive variant of hotkey that handles open/close functionality (e.g. <Ctrl+Hotkey>)
3. The cursor is repositioned within the original function editor pane

   If resulting from **UC3**, cursor defaults to called function's location in original function

   If done via mouse click, cursor should reposition to wherever mouse identifies, as is standard with text editors

   If done using a hotkey, cursor defaults to called function's location in original function

4. End

Postconditions:

1. None

UC5: Switch to called function's contents

Preconditions:

1. **UC1**
2. **UC4** (unless resulting from **UC1**)

Use Case Scenario:

1. If resulting from **UC1**, jump to step 3
2. User uses one of the following methods to switch to called function's contents:

   Standard mouse click to switch to inline editor pane

   An intuitive variant of hotkey that handles **UC1**, **UC3** functionality (e.g. <Ctrl+Hotkey>)
3. The cursor is repositioned within the inline function editor pane

   If resulting from **UC1**, cursor defaults to called function's declaration within inline editor

   If done via mouse click, cursor should reposition to wherever mouse identifies, as is standard with text editors

   If done using a hotkey, cursor defaults to called function's declaration within inline editor

   If **UC5** has been previously executed and development time allows, cursor should be returned to last location within inline editor
4. End

Postconditions:

1. None

Chapter 6

Sequence Diagrams

At this stage of development, sequence diagrams will be expressed at a conceptual level. Several preliminary sequence diagrams are shown below to illustrate the most crucial use cases.

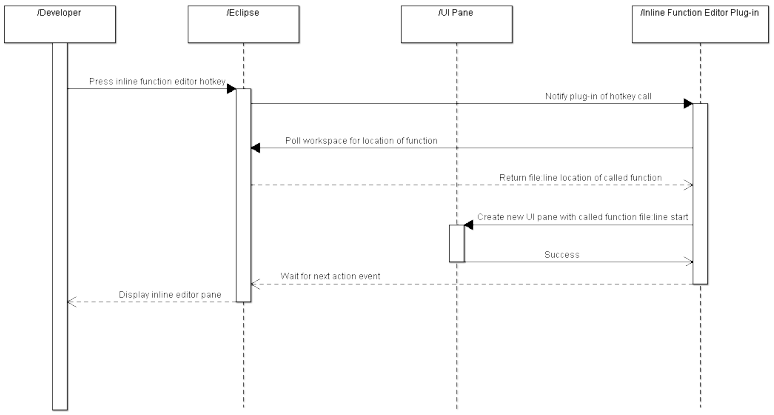


Figure 6.1: Open inline function editor

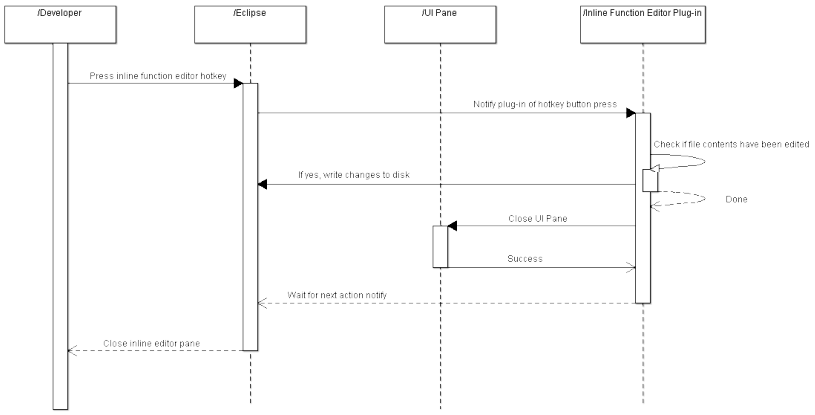


Figure 6.2: Close inline editor pane

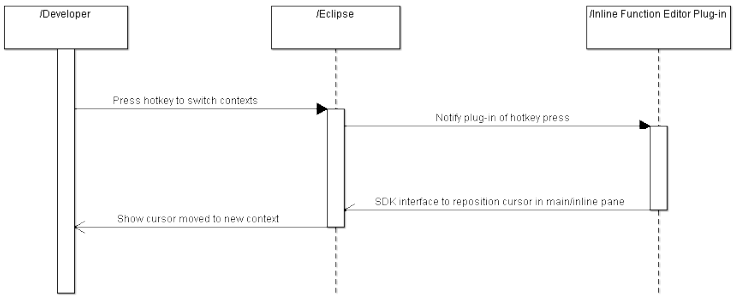
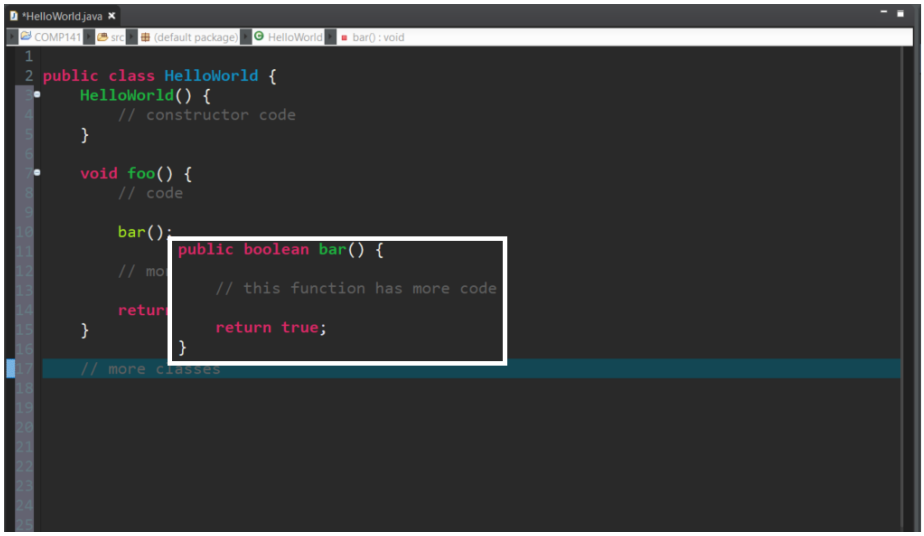


Figure 6.3: Switch editor pane contexts

Chapter 7

UI Design

Below is an initial mock-up of our design. Given the architecture of Eclipse, creating a pseudo-inline editor pane tied to the location of the called function is the current path of least resistance, but this may be subject to change as the development continues. Implementing a literal inline function editor like Visual Studio would require the functionality be built into the editor panes, which goes beyond the scope of a plug-in developed over a few months. Creating an individual instance of a pane with the file opened to the declared function should still take the same number of steps as an inline function editor like Visual Studio's.



Chapter 8

Glossary of Terms

Eclipse

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment (*Eclipse (software)*, n.d.).

Hotkey

A keyboard shortcut (e.g. <F3>).

Inline Function Editor

An instance of an editor that is opened "inline" with the original text. See the chapter "UI Design" for graphical aids. For the purposes of this RAD, an inline function editor is considered an editor pane contained within the originating editor pane.

Integrated Development Environment (IDE)

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Most of the modern IDEs have intelligent code completion (*Integrated Development Environment*, n.d.).

Open-Source

Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose (*Open-Source Software*, n.d.).

Pane

A section of a window that provides the user with additional information or quick access to features commonly used in a software program (*Pane*, n.d.).

Plug-in

In computing, a plug-in (or plugin, add-in, addin, add-on, or addon) is a software component that adds a specific feature to an existing computer program (*Plug-in*, n.d.).

References

Downloads and tools for windows 10. (n.d.). Microsoft. Retrieved from https://developer.microsoft.com/en-us/windows/downloads

Eclipse foundation. (n.d.). Eclipse Foundation. Retrieved from https://www.eclipse.org/documentation/

Eclipse (software). (n.d.). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Eclipse_(software)

Integrated development environment. (n.d.). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Integrated_development_environment

Open-source software. (n.d.). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Open-source_software

Pane. (n.d.). Computer Hope. Retrieved from https://www.computerhope.com/jargon/p/pane.htm

Plug-in. (n.d.). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Plug-in_(computing)