Keeping Functions Inline

Mark Fraser
m_fraser3@u.pacific.edu

Jillian David
j_david1@u.pacific.edu

February 2, 2019

Contents

Chapter 1

System Overview

1.1 TODO: Basic Architecture

1.2 Motivation

Software engineers' and programmers' services alike are in constant demand in today's technologically centered society. Because of there is insufficient supply to meet demand, software development personnel command higher salaries. Thus, from a company's perspective, SEs and programmers should be supported as much as possible to maximize their efficiency. From the SE's and programmer's perspective, software development should be as seamless as possible to facilitate implementing their code logic. Many programmers develop in Integrated Development Environments (IDEs) to leverage shortcuts and plugins to streamline their workflow. Eclipse is one of the most popular open-source IDEs with many plugins, 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 plugin 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 Implementations

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 TODO: Visual Studio

2.3 TODO: IntelliJ

Chapter 3

Project Stakeholders

3.1 Product Owner

The owner of the plugin. This stakeholder is responsible for delivering a product that meets the standards of Eclipse plugins and can be added to the plugin store. This stakeholder is also responsible for maintaining the integrity of the code base.

3.2 Users/Clients

There is only one user/client of this plugin, and that is users of Eclipse. 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 plugin will use it with the intention of increasing their development efficiency through better code navigation than what Eclipse currently offers.

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 plugin must inherently be stable to achieve this requirement.

4.2 Keyboard Shortcuts

Developers are most efficient when developing on the keyboard, as keeping the hands in one place and using the same interfacing tool maintains a consistent context. This is opposed to switching between using the keyboard and mouse, or keyboard and screen, or all of the above. Thus, a plugin dedicated to efficiency should therefore provide some hot key functionality on the keyboard to best fulfill its goal.

4.3 Menu Options

For developers who do not wish to bind this plugin's functionality to hot keys, they must still be able to interface with the plugin through the Eclipse menus. Therefore, menu options must be implemented to guarantee interfacing functionality.

4.4 Efficient Content Display

The plugin'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

Not all 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 plugin should be architected to maintain functionality and require minimal changes when ported to future versions of Eclipse.

Chapter 5

Functional Requirements

5.1 TODO: Use Case Diagram

5.2 TODO: Casual Use Case Descriptions

5.3 TODO: Fully Dressed Use Case Descriptions

Chapter 6

Sequence Diagrams

Chapter 7

UI Design

Chapter 8

Glossary of Terms

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.[1]

[1] https://en.wikipedia.org/wiki/Integrated_development_environment