Logo, company name

Description automatically generated

**CSCI 5306 – Applied Program Comprehension**

Instructor – Dr. Tami Meredith

Assignment – 2

*Crew – 11*

1. Benny Tharigopala – B00899629
2. Hardee Garala – B00869195
3. Heli Patel – B00868906
4. Parthkumar Patel – B00899800
5. Vishal Jaiswal – B00867181
6. Vishvesh Naik – B00885077

**TO DO:**

**Danny:**

Rephrase Introduction and Why use PC tool

Window Features (3) from Source Insight

Experience with source Navigator

Experience with Source Insight

Conclusion – **Why Source Insight over Source Navigator?**

**Vishvesh:**

Comparison with other tools

Experience with Source Insight

2-3 limitations of source insight

Conclusion

****

**Hardee:**

Introduction

Why PC Tool

Limitations

Experience with Source Insight

Conclusion



**Heli:**

Introduction

Why PC Tool

Limitations

Experience with Source Insight

Conclusion



**Parth:**

Introduction

Why PC Tool

Limitations

Experience with Source Insight

Conclusion



**Vishal:**

Introduction

Why PC Tool

Limitations

Conclusion



**Introduction**

**Danny:**

The purpose of this report is to evaluate a Program Comprehension tool and describe its features, shortcomings, and other properties. Large legacy software can typically last more than a decade, resulting in many maintenance challenges. Furthermore, the constantly changing nature of the software industry makes not only the size of the software grow, but also results in an increase in the number of programmers. Over the course of the extended development process, original specifications and design intentions can be lost, the documentation can become unreliable, and code quality can be compromised. Consequently, methods and tools for newcomer programmers on how to better understand existing code (or at least relevant modules) and reduce the time until they can successfully integrate into the industry have become increasingly prevalent.

**Why use a program comprehension tool?**

**Danny:**

Keeping large, legacy software up to date often results in increasing development time and costs due to the size and complexity of the codebase and its documentation, their continuously deteriorated quality and the fluctuating number of developers. Code comprehension tools address the issue by providing different textual information, graphical views, and metrics on different abstraction levels. In addition to analyzing the codebase, these tools usually examine the build information, the version control repository, and other information sources. Source code editors and integrated development environments (IDEs) are not effective at code comprehension because they are optimized for writing new code, not for browsing. During development, this can lead to frequent switching between environments, hindering effective programming and boosting development costs. The primary objective of a program comprehension tool, therefore, is to help understand large legacy software systems.

**What does SOURCE Insight offer?**

Source Insight, primarily a code analyzer and code editor, helps developers assimilate a code base before performing any maintenance activities or feature updates. Since its critical for programmers to understand what they are working on before performing actual code change(s), Source Insight facilitates an efficient and diligent workflow by identifying where objects of interest are used and rendering relevant call trees for users, through the use of interactive and cohesive data panels. Using this information programmers can determine the possible costs associated with any planned updates and make informed decisions. The tool parses the input source code and offers users the capability to traverse through items of interest, seamlessly, with mere mouse-clicks. The tool is well-suited for Large Scale systems with millions of Lines of Code since its powerful and versatile with features like Syntax Formatting, Smart Renaming and Object Referencing. A noteworthy feature of Source Insight is that after it parses the source code it dynamically maintains its own “Database” of information by associating data with symbols and presents useful contextual information without any prompt.

**…**

**…**

**…**

Here’re some of the features offered by Source Insight:

**Project Window:**

The project window is a container which has tabs for all the project-related panels. It contains tabs for the following set of Information:

A project symbol list, where one can view all the symbols associated with the source code, Typical Examples of symbols include Methods, Variables, Import Statements, Macros and “typedefs”. A project file list, which lists the entire set of files that compose an application. This section includes information about the path to a file, its size, the last modified date and Code Metrics such as, Lines of Code and Decision Count.

Graphical user interface, text, application

Description automatically generated

A project symbol categories section, which is similar to the Symbol List, but instead groups the objects in a project by their category. Finally, a project folder browser which offers a File Browser view of a projects directories.

Now let us consider an example, referring to one of our goals in Assignment 1, “*How are we going to define the physics of the oil slick?*” Without the tool, users would have to preform a grep search in a file browser for files that are relevant to “physics”. However, with Source Insight, users can input a keyword and files which are relevant to the keyword are rendered in the Project window. From Figure 2, we can observe that files which have the literal “physics” in their file paths are also displayed in the window!

Table

Description automatically generated

**Context Window:**

The context window, as its name suggests, provides contextual information with respect to an object that is currently being accessed or edited. The context window can be considered as an extension of other panel windows, as it enhances the functionality of the other windows. For example, the context window opens files selected in the Project File List tab and navigates to occurrences or definitions of symbols when a user selects a symbol from the Project Symbol List tab. What’s more, users can customize their preferences, from Fonts to Automatic Tracking (which instructs the Context Window to look up objects according to the position of the cursor.

As an example, lets consider another goal from Assignment – 1. “*Investigate how the code for the existing features is implemented.”* Let’s assume that we found our foothold for this goal. We have determined the set of files where features of interest are defined. However, we need to determine what, where and how variables, associated with features, are defined. Suppose the requirement states that “A kart which is a victim of an oil slick must encounter a collision. We have located the function “*createCollisionShape()”. However, we need to determine the point of declaration of one of the parameters “m\_collision\_shape”. The context window is particularly useful in this scenario since users just need to click on the variable and the window will automatically navigate the current view to the point where the variable was initialized.*

Graphical user interface, text, application

Description automatically generated Graphical user interface, text

Description automatically generated

**Relation Window:**

The relation window, in Source Insight, displays relationship between a selected object and other objects. Besides showing function call trees, class hierarchies and structure members, the Relation Window works in the background, automatically tracking what a user select and showing relationship information accordingly. One of the common usages of the Relation Window is to display the Class Inheritance information to users. The feature dynamically records relationships between objects, as updates are being made to the source code and allows users to refresh the relation window to recompute relationships.

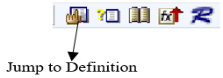
The window offers options to view outlines of relationships (just the participating objects, such as member functions), to view relationships through a Horizontal Graph (that grows left to right) or a Vertical Graph (that grows from top to bottom). To illustrate this feature, lets consider an example, referring to one of our previously established goals, “*How will we add the feature oil slick to the battle mode and change the methods of class* ThreeStrikesBattle?”. Before starting to make changes to the class ‘ThreeStrikesBattle” developers should review the relation ship of the class with other classes/objects to ensure that the changes that will be made do not have an adverse impact on other classes and also to determine where else any form of associated change should be made to enable the new functionality.

Graphical user interface, text, application

Description automatically generated

**Relation Window:**

Jump to Definition:

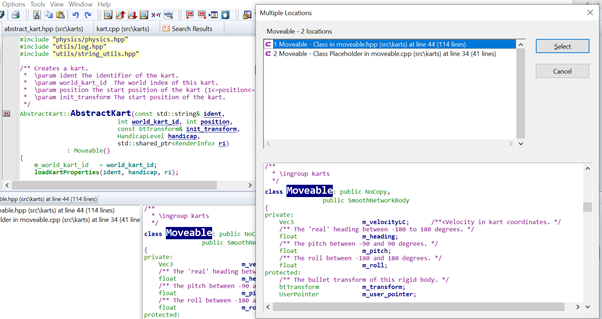


The Jump to Definition command navigates to the symbol under the cursor's declaration. The Jump to Definition command locates the symbol from the very first word in the current selection, opens the symbol's definition, displays it in a window, and selects the symbol name. If your project has many symbols with the same name, Source Insight will prompt you to choose which one you want. Jump To Definition make it simple to go through a method call chain.

To use this command:

* Make a selection inside the symbol name as it appears in the source file.
* To retrieve the actual symbol definition, use Alt+= or Ctrl+double-click with the left mouse button.

Here, the AbstractKart() function calls the Moveable() method. To learn more about what the method does or the purpose of the Moveable method, we can utilise Jump to Definition, which will take us to karts.cpp, where it is defined. As a result, it will be simple to comprehend. We may also access a file by clicking on its name in a #include statement and using the Jump to Definition command.



Symbol Info:



Symbol Info is one of the several built-in internal macro functions in Source Insight. The Symbol Info command opens a pop-up window that displays the definition of the symbol currently under the cursor. This is a simple way to check an identifier's definition.

There are different parts of the Symbol Info window, Symbol Name, Type, and Location of the symbol, as well as the file and line number, are displayed at the top of the window.

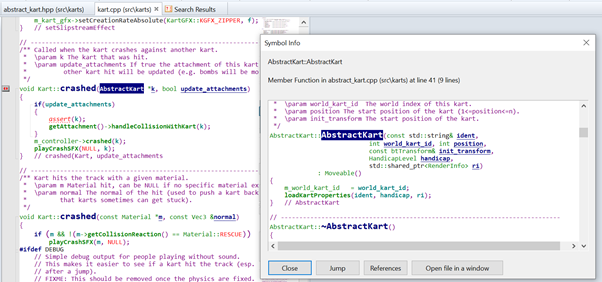
The top left of the dialogue box, below the symbol name, displays the source file name and line number where the symbol is defined. The symbol's size in lines is also shown.

The contents of the source file where the symbol is specified are displayed in this scrollable pane/Text Area. Jump Click here to exit the Symbol Info box and go directly to the symbol definition.

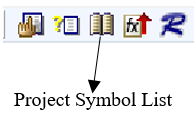
Click References button to look for references to the symbol across the project.

When you edit files in the project using your source control application, Source Insight automatically notices and progressively updates the symbol information for you. Symbol data is saved on disc in an indexed symbol database that is essential to the project.

When the pointer was on the AbstractKart symbol, we could utilise Symbol Info to learn more about it.

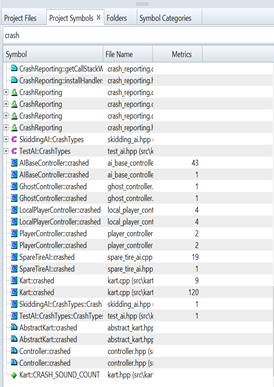
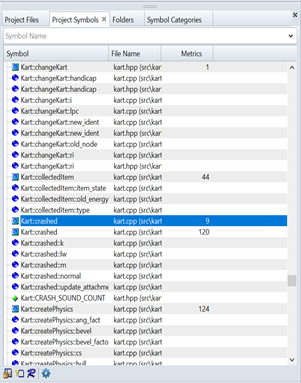


Project Symbol List:

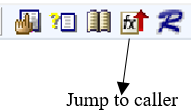


In the Project Window, this command displays all project symbols. The Project Window, unlike the modal dialogue box, is modeless, and the Context and Relation Windows will be captives to it.

When we click on the project symbol list, it will reveal all of the symbols that are present in the project STK. We may also search for a certain symbol to see how many symbols with the name are available.



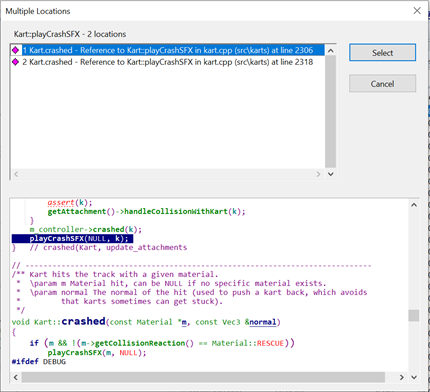
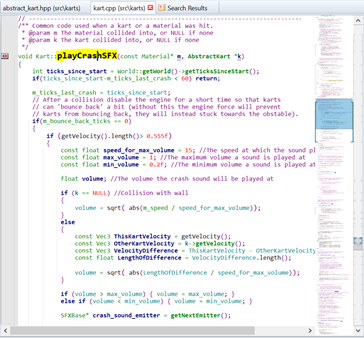
Jump To Caller:

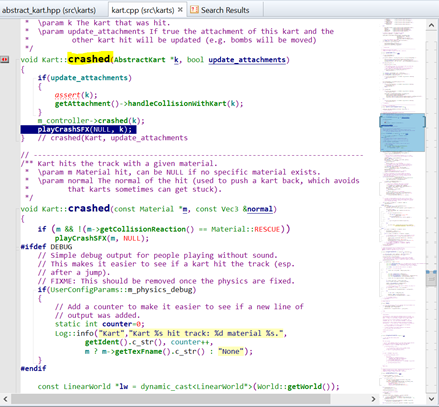


If any, jumps to the caller of the specified function. For instance, if you place the mouse on a function name and select Jump to Caller, you will be sent to the function that calls it. If it is called by more than one function, you will have a list from which to choose.

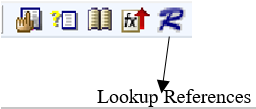
To perform this command, right-click on the symbol and select Run as a shortcut from the shortcut menu. This only works if the object beneath the cursor is a function name.

We have a definition of playCrashSFX here, but to further understand the project's function and functionality, we need to know where and how this method is called. When we click on the Jump to caller tool, we will be presented with two possibilities from which we may select the one we wish to explore.





Lookup References:

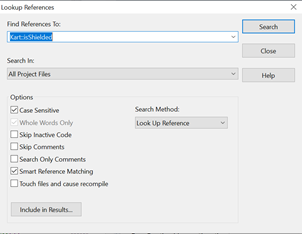


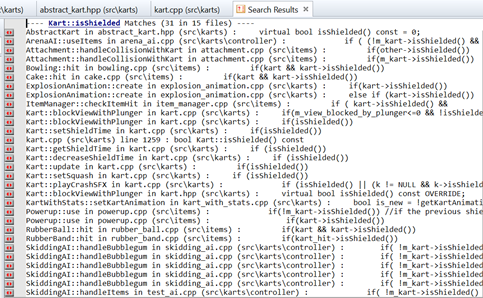
The Lookup References command looks for references to a specific symbol in the current project. Source Insight employs symbol indexes to speed up searches. All source code material, including comments, contains references. You may, however, choose whether or not certain locations are searched.

There are different parts in Lookup References dialogue box

For example, click inside “IsShielded()”, and then click on the Lookup References command, and Source Insight will open a Search Results window, which lists all the places that reference “IsShielded()” in your project. Then you can select one of them and it will display search result as shown in the umage.

When you utilise Lookup References from a symbol dialogue box or window, Source Insight stores the exact symbol references along with this text field, or you may type the symbol name in the Lookup reference box. Based on the context of the cursor point, Source Insight will determine the specific symbol instance you require. Search In drop-down list contains a list of document type. You may use this list to narrow your search to a certain file type or only the current file. Lookup References will be picked from this list as Search Method. Source Insight will also detect alternate ending versions of the keywords you supplied if smart reference matching is enabled.





**SOURCE Insight Limitations**

**SOURCE Insight experience**

Source Insight can be termed as a powerful tool that appropriately addresses the needs of users and assists code base maintainers to perform changes or updates without difficulty. The tool parses source code to the most granular level (objects in this case) and helps developers understand an existing code base with relative ease since it evaluates the cost of making updates by recognizing usages of relevant functions and objects. The tool has unique innovations like the Relation Window and Context window, through the use of which, programmers can seamlessly navigate and edit code and jump to calls or references of various objects. Another feature that we found to be extremely useful is the alternate text that pops up whenever users hover over a particular feature! The text provides comprehensive information about the feature and is especially helpful for novices.

Graphical user interface, application

Description automatically generated

Another key benefit would be the capability to preview function and class definitions, excluding the painstaking process of opening/closing files and cluttering the workspace. In our opinion, this helps save time and allows code maintainers to work efficiently and diligently. Call graphs, class tree diagrams, advanced editing features and dynamic information panels, all, when used efficiently, lead to a more productive workflow, and make the tool a perfect choice for getting up to speed and understanding a new code base.

## SOURCE NAVIGATOR

**What does SOURCE-NAVIGATOR offer?**

**Feature to view and filter various object types in an application’s source code directory:**

Graphical user interface, application

Description automatically generated

**Navigate a project directory through one of the following options:**

Classes

Methods

Functions and

Files

Text

Description automatically generated

**Column Filter:**

Graphical user interface, text, application

Description automatically generated

**Viewing Dependencies:**

Graphical user interface, text, application, email

Description automatically generated

Clicking on a file will open its corresponding code in the tool. Similarly, clicking on a method/Function under a file will open the file and automatically navigate to the code associated with the Function.

Graphical user interface, text, application, email

Description automatically generated

**Include Browser:**

The Include Browser assists users to view what source files are imported by the current object in question. In essence, it displays relationships which can be defined by the terms “Include” and “Included By”. The following figure illustrates how a file – “addon.hpp” includes files such as “assert.h” and is also included by files such as “addons\_loading.hpp” and “kart\_properties.hpp”.

The number of levels, based on which the relationship extends to, can be controlled by the option – “Levels”.

Graphical user interface, application, Word

Description automatically generated

**Class Hierarchy Browser:**

The class hierarchy browser helps users identify the entire class hierarchy, base classes, and super classes with respect to a specified class object. For instance, Source-Navigator recognizes that the class “Widget” inherits from the class “SkinWidgetContainer” and the class “KartStatsWidget” inherits from “Widget”.

Graphical user interface, application

Description automatically generated

**Grep:**

The Grep function allows users to input regular expression(s) and view code associated with the expression(s). In addition, users can filter the files to limit their search; for example, entering \*.java restricts the search to only Java files. Clicking on an item in the Grep window opens the file in a file editor, with the cursor positioned over the selected line. Grep results can also be step-through or step over. To filter your Grep results, use the Format combo-box to select an option. The feature also facilitates search in multiple directories and saves results of multiple searches, for users to review later.

Graphical user interface, text, application, email

Description automatically generated

**Retriever:**

This feature assists users to search for patterns corresponding to symbols, in the database. For example, users can enter a pattern related to any word containing the expression “track” with the pattern “\*track\*”.

Graphical user interface, table

Description automatically generated with medium confidence

Clicking on an object, say, “addTrack(md)” would open the item in the editor and displays the object in context of the source code.

Text

Description automatically generated

**Class Browser:**

The Class Browser displays a holistic view of class objects their methods, attributes and return types in addition to the parameters linked with each method.

Graphical user interface, text, application, email

Description automatically generated

A picture containing timeline

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**History:**

Features like history are one of the most important/essential features in any tool or software to observe our previous activity and actions. We need a history tab to track our steps and activity like a web browser. In the source navigator, we have a history feature with additional functionality that includes previous features like Retriever, Grep, Edit, Hierarchy, Class, Xref, Include. All these features inside of history correspond to its history, respectively. For example, in Edit, we can view which files were edited after SN was open. Furthermore, a back and forward button also perform a similar task where we can perform a quick back and forward task.



Figure 1: View History - Forward and Backward Button

Graphical user interface, text, application, email

Description automatically generated

Figure 2: History Feature

Parth

**Why PC tool is Important?**

A software developer uses the source code of the software as the primary reference of the activity to try to comprehend a software system and its behaviors as part of program comprehension. In the software development lifecycle, software comprehension has always been crucial. Program comprehension tools let the software developer create and develop all the necessary understanding during the software maintenance process. They make it simple to comprehend and explore the current code base by providing cross-references, overviews, and visualization. Although the developer is mostly responsible for program comprehension, program comprehension tools assist the developer in determining what is required and, thus, what to read.

**Advantages**

* Allows for the use of queries for code search and the creation of user-defined rules.
* Aids in determining the project code's underlying architecture.
* Using the diagramming feature, you may quickly and easily recognize complex code.
* Control and data flow analysis
* Relationships and dependencies are easily visualized with this tool.
* Color is used effectively to bring attention to and spotlight methods.

**Limitations**

* When method was viewed in the Code Editor, but it did nothing. This was a missed opportunity to make these features far more useful, and it made it difficult to jump between methods fast while looking at the code.
* There is no simple way to progress up a level, to another class, or to return to the previous one if you lose your place in the Code Editor as File Explorer. You must locate it or conduct a name search.

**Testing – Determining success of changes**

**What kind of testing is being used, where is the test data coming from?**

When a programmer adds a new feature to the SuperTuxKart project, testing is crucial. Since this is a multi-player game. If the code implemented by the programmer as part of the new feature breaks the game's existing functioning, it will have a negative impact on the game, and end users will raise issues. It's also possible that gamers will cease playing the game, resulting in a loss for the SuperTuxKart creators. This emphasizes the need of testing before merging new code into existing code. Considering the goal, the programmer is attempting to locate the testcases and types of test data used to test the SuperTuxKart project to test the newly added code. Finding the exact position of the testcases within SuperTuxKart's massive project is difficult for a programmer working on the project for the first time. The programmer may have to spend a significant amount of time locating the exact spot where the testcases for the project's functionality are written. Sometimes programmers must rely on coworkers who have previously worked on the project. To prevent wasting time, the programmer should analyze the files that contain the project's testcases using program comprehension tools. The programmer can simply complete such operations with the Source Navigator tool. The programmer can use this tool to not only detect the project's test methods, but also to identify the project's code coverage data. The programmer can utilize the code Editor function of the tool to find the project's test methods. From the specified methods, the programmer may quickly discover the test methods and check the relevant method.

**Compare/contrast of other tools**

|  |  |  |
| --- | --- | --- |
| **Tools** | **Features** | **Languages Supported** |
| **CodeQl** | * Control and data flow are examined * Exploring threat models | C, C++, Java, JavaScript, TypeScript, .NET, Python, Go |
| **SonarQube** | * Aids in decreasing code complexity * Detects code vulnerabilities * Identifies code repetition * Display metrics to reduce maintenance costs * Error detection * Plugin available for SonarQube platform | C, C#, C++, Java, JavaScript, TypeScript, VB.NET, Python |
| **SourceMeter** | * A precise and in-depth static analysis tool * Identify rule violations especially relevant to Android * Checks the code standard rules * Identify the code repetition metrics * Scans the runtime exceptions inside the JAVA code * Checks the code metrics such as complexity, coupling, and cohesiveness | C, C++, Java, Python |

**Experience with the tool**

**Source Navigator - Abbreviations**

cl: classes

con: Constants

e: Enums

ec: Enum Values

fd: Function Declarations

fr: Friends

fu: Functions

gv: Global Variables

iv: Instance Variables

ma: Macros

md: Method Definitions

mi: Method Implementations

t: Typedefs

un: Unions

lv: Local variables

ud: Undefined

**Cross-Reference:**

r: Read p: Passed w: Written u: Unused

v: Virtual

s: Static

+: Override member of base class

-: Overridden in subclass

**conclusions**

**references**

1. Z. Porkoláb, T. Brunner, D. Krupp and M. Csordás, "CodeCompass: An Open Software Comprehension Framework for Industrial Usage," *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 361-3618.
2. Z. Porkoláb and T. Brunner, "The CodeCompass Comprehension Framework," *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 393-3933.