

# **DevAssist**

**Developer Assistant for VisualStudio**

by

**H.D.L. Dayarathna**


149207B

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfilment of the requirements of the Degree of Master of Science in Information Technology.

**June 2017**

### Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student	Signature of Student
H.D.L. Dayarathna	

Date: 23/06/2017

Supervised by

Name of Supervisor	Signature of Supervisor
Mr. Chaman Wijesiriwardana	

Date: 23/06/2017

## **Acknowledgement**

There are many important individuals who supported me to make this project a success. I would like to extend our sincere gratitude to all of them. My deepest gratitude and warmest appreciation goes to Mr. Chaman Wijesiriwardana, my supervisor for his valuable support and guidance throughout the development of this project. His advices and suggestions were immensely helpful in both designing and implementation phases, to develop a better and more value-added system at the end.

I am also grateful to all the authors of the reference materials we have used throughout this project and the web sites that we used in gathering data. Finally, I wish express a sense of gratitude to my family and all my friends for their support, strength, and help for completing this research this extend successfully.

## **Abstract**

With the rapid growth of the information technology, large number of programmers enter the software development field every year. Having tough deadlines in their day to day task completion, the performance monitoring for career growth, the inner contest is higher and the availability for peers is less. Due to higher complexity of the software components, the developer face lack of technology knowledge and domain knowledge every phase when solving deep logics. DevAssist is the best solution to feed knowledge into developer's mind. It is always better to have a method to guide developers in programming tasks acting like an experienced peer especially considering busy schedule of developers.

DevAssist plugin is an efficient solution that will allow developers to be aware of the updated technology as well as domain strategies. Considering business perspective, any software company may satisfy if there are many exceptional developers there. As developer, anybody would like to have someone as a peer overlooking your shoulder and providing guidance. So DevAssist is a way to have positive responses for own desires.

Basic idea of implementing such a system is to, make it easier to solve complex logical problems by suggesting solutions to the developers. Here it uses fewer inputs for the system and system will run as a background process. This will help users of the system to continue their focus smoothly as user doesn't have to bother on saving the useful content time to time manually. In this effort application is fed by the inputs automatically and then it detects user activities.

# Contents

Introduction.....	1
1.1 Prolegomena.....	1
1.2 Background & Motivation .....	2
1.3 Problem statement.....	4
1.4 Hypothesis .....	5
1.5 Aim and objectives .....	5
1.6 Solution approach .....	6
1.7 Structure of the Thesis .....	7
1.8 Summary.....	7
Review of Others Work .....	8
2.1 Introduction.....	8
2.2 An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks .....	8
2.3 COSME: A NetBeans IDE plugin as a team-centric alternative for search driven software development.....	8
2.4 SurfClipse: Context-Aware Meta Search in the IDE.....	9
2.5 Prompter: A Self-confident Recommender System .....	10
2.6 Comparison of other approaches to DevAssist. ....	11
2.6 Summary.....	12
Usage of Technology .....	13
3.1 Introduction .....	13
3.2 Technologies Used .....	13
3.3 Microsoft technologies .....	14
3.3.1 What are Microsoft Technologies?.....	14
3.3.2 .NET Framework .....	14
3.3.3 What are the components of .Net Framework? .....	14
3.3.4 Why .Net Framework?.....	15
3.4 Levenshtein distance .....	15
3.5 Summary .....	16
An Approach to DevAssist: Developer assistant for VisualStudio IDE .....	17
4.1 Introduction.....	17
4.2 Hypothesis.....	17
4.3 Input to the system.....	17
4.4 Output of the system .....	17

4.5	Users of the system.....	18
4.6	Process .....	18
4.6.1	Query Building Process .....	18
4.6.2	Searching process .....	19
4.6.3	Ranking model .....	20
4.7	Features .....	22
4.8	Summary .....	22
	Analysis and Design .....	23
5.1	Introduction .....	23
5.2	The Proposed Design .....	23
5.3	Design Diagram.....	26
5.4	Detailed description about the Design Diagram.....	26
5.5	Sequence Diagram of the System.....	29
5.6	Summary .....	29
	Implementation .....	30
6.1	Introduction.....	30
6.2	Overall solution.....	30
6.3	Implementation of Query Builder Service .....	30
6.4	Implementation of Search Service .....	39
6.5	Implementation of Ranking Model .....	41
6.6	Summary .....	47
	Evaluation .....	48
7.1	Introduction .....	48
7.2	Evaluation of the prototype.....	48
7.2.1	Generate search query .....	48
7.2.2.	Ranking results.....	50
7.2.3	Accuracy of receiving a result using DevAssist plug-in .....	51
7.2	How our solution differs from the others' work.....	52
7.3	Summary of the entire report.....	52
7.4	Summary .....	53
	Conclusion & Further work .....	54
8.1	Introduction .....	54
8.2	Objectives .....	54
8.3	Achievements of objectives .....	55
8.4	Problems Encountered .....	55

8.5	Further work .....	56
8.6	Summary .....	56
	References.....	57
	Appendix.....	59

## List of Figures

Figure 5.1: Module that captures data manually from the user .....	24
Figure 5.2: Module which gathers data from code editor .....	25
Figure 5.3: Design diagram of the system .....	26
Figure 5.4: Sequence diagram of the system.....	29
Figure 6.1: Screenshot of Visual Studio IDE with DevAssist.....	31
Figure 6.2: Levenshtein algorithm example.....	42
Figure 7.1: Query generation example 1 .....	48
Figure 7.2: Query generation example 2 .....	49
Figure 7.3: Query generation example 3 .....	49
Figure 7.4: Ranking result 1 .....	50
Figure 7.5: Ranking result 2 .....	50
Figure 7.6: Ranking result 3 .....	50

## List of Tables

Table 2.1: Comparison between other's work and our solution .....	11
Table 6.1: C# keywords .....	33
Table 6.2: Contextual keywords in C#.....	34
Table 7.1: Accuracy of received results using DevAssist plug-in .....	51
Table 7.2: Evaluation of DevAssist against Related wok.....	52



# Chapter 1

## Introduction

### 1.1 Prolegomena

Software development is a tedious task that requires extensive knowledge different technologies. Many programming languages and technologies emerge every day. It is a stressful, time consuming task for a developer to keep up with new technology and to make use of them when needed. Even though the technical knowledge is up-to-date, it may be not enough when working with the industrial projects. They must earn vast domain knowledge in addition to knowledge of the technologies. Developers must deal with unknown or partially known parts of a system. At the same time developers, should go beyond the knowledge that they already possess.

Most modern software development relies on reusable software asserts like frameworks, libraries which are exported through application programming interface (API). These are large and heavy structured information spaces that the developer must understand and navigate to complete their tasks. But the stressful deadlines are compulsory in the software field. At the moment developer is an isolated person although he or she is a part of a large development team. In other words, developer must do his/her own research when confronted with any technical/non-technical issue.

Thanks to many sites and information in the web, developers are able to learn and update knowledge faster to support today's growing software requirements. Therefore, the approach to software development has become more a search driven approach. This has lead developers spending more time in searching proper code samples than actual time spent on programming itself.

To aid these search-driven development approaches many technology specific documents, sites and forums are available in the Internet. Few of the most well-known forums are Stack Overflow, Snipplr and Github. Most of these forums are community based and require help from search engines such as Google to retrieve the relevant documents from

them. However, programmers should go through several forums and refine their search many times in order to find a proper sample code or an explanation. This can be time consuming and will contribute to many search hours in software development projects.

In addition, most of the developers heavily depend on integrated development environment (IDE) for development. There are many IDEs being used by different developers depending on their development language and requirements. Few of the mostly used IDEs are Visual Studio and Eclipse. Eclipse is a free open source IDE while Visual Studio is Microsoft's proprietary solution mainly used for .Net development.

These IDEs are well equipped with many features and functionalities to assist developers. There is a great deal of help given by IDE when concerning programming language commands, testing and deployment. The IDEs that developers currently use are highly advanced in their kind and equipped with many tools for successful software development, testing, launching and even repository management. However, to date IDEs do not have any code snippet search help. Code snippet search tool, which can prompt solutions by understanding the context of the current code being developed, can significantly increase the effectiveness and the performance of a developer.

## **1.2 Background & Motivation**

The development environment for software includes all the development related tools. In addition, large application programming interfaces (API) include extensive documentations, reference documentations, user manuals and code examples. As an experienced developer, the facilities provided by integrated development environment (IDE) [1] are useful. But integrated development environment (IDE), information spaces and other dynamic knowledge resources provide unfiltered solutions regarding technology. However, when there are issues related to the logic, or issues related to the domain, the integrated development environment (IDE) will not be a player anymore.

The developer is merely isolated person even though he is a part of a bigger software development team. Thus when a developer faces situations like domain issue, logic issue and also may be a technology related issue, collogues, online help forums [2], mailing lists [3], blogs, Q&A websites will help randomly. Other resources may not provide the optimum solution. The solution may be outdated, has performance issues or less secure or unethical. As a developer, the solution cannot be analyzed to check whether it is the best without additional peer reviews, extensive logic review or implementing it.

The coding speed of a developer directly affects to the deadline and ultimately for the performance of the developer. Referring the provided frameworks, libraries by integrated development environment (IDE) will be time consuming. On the other hand, the developer must have a structured knowledge and extended practice to apply these technology adapters to own solution. The deadlines are strict and must be strict hence the nature of the software field. Therefore, when a developer gets stuck in a critical task, it is not possible to patch a solution by learning a new technology from scratch completely. In these cases, most of the developers just learn a specific part of a technology that is required to fix the critical task they must perform.

Apart from that, many new junior developers enter the software development field every year. They gain knowledge by collecting experiences. Beginners will come across many technical issues, which their senior peers have already come across but they cannot depend on experts always or reach out to them easily. Due to these reasons, developers always tend to maintain and update Q&A sites such as Stack Overflow with problems they faced previously so the beginners don't have to reinvent the wheel. As beginners, they should learn to manage time, get up-to-date with new of technologies, collect domain knowledge etc.

However, it would be nice to have tool, which perform like a peer developer with more experience that will always looking over your shoulder and analyzing the code in progress. Expert knowledge is available online with Q&A sites and forums. Nevertheless, what is lacking to automate the process of peer developer is a tool, which can analyze the in-

progress code snippet, query web for solutions and rank and prompt it in an IDE. So, the motivation was to suggest a solution, which is a tool embedded in IDE, which act like a peer. The solution should be available only when the developer needs it. It should behave like a peer by providing solutions to developer's problems, guiding the developer, collecting vast accurate information available in the web, filtering best solutions, optimizing the written source code, be available in the currently working integrated development environment (IDE).

The suggested system can be developed further, if it can connect with many integrated development environments (IDE). If a system acts like a peer, like a mentor, it will be the best tool, which can be used by a developer to gain knowledge. In addition, this tool will increase the productivity of the developer by decreasing the time spent on searching best solutions.

The suggested system will be intelligent enough to generate the query and rank the most suitable solution. However, there is much inaccurate information in the Internet, which can act as a match to the generated search query. To achieve accuracy, we can consider some well-known accurate resources that are popular among developers. Basically, Stack Overflow [9] which is the largest online community for programmers to learn and share their knowledge can be used as a main information resource for the system. Apart from that, the user will be facilitated to connect with the other standard online communities, which are some Q&A web sites, forums etc. This tool will become one of the most useful tools included in the IDE when the algorithms used can precisely point out what developer is looking for.

### **1.3 Problem statement**

Developers tend to use tremendous time searching for better solution over various resources with respect to technology or logic. The issue, focused in this research is how an automated system can provide better assistance to software developers similar to an expert peer programmer. Although the problem of time spent on updating knowledge, find

solutions are common in the software field; the proper automated solution has not yet been initialized.

In summary, the problem is lack of a proper navigator system for software developers. Research related to this problem area cannot be seen much. Since the proposed solution must be an intelligent system, which must possess the knowledge beyond an experienced developer, the percentage of unsolved areas of the problem may be higher.

Among many challenging areas of this problem, most challenging would be to predict accurate solutions to prompt to users and further improve its algorithms in order to provide a better solution by understanding the context of the code. This will be a challenging task and would need constant upgrades to the tool.

Fine tuning the intelligence of this system is a challenging in this research. In the same time, the design must concern about the high performance, security, robustness, and technical barriers that will come across implementing the solution.

#### **1.4 Hypothesis**

The hypothesis of the research is an enhanced way to assist developers by prompting performance issues, best solutions or simply technical documentation for code using a tool embedded in the Integrated Development Environment (IDE). This tool will generate search queries based on the code context and pass it to API services of search engines and top technology related forums. Extracted solutions are then evaluated and suggest available help by the relevance to the problem considering various decision-making criteria.

#### **1.5 Aim and objectives**

The aim is to develop an intelligent plugin to the Integrated Development Environment (IDE), which will support as a guider as well a peer to software developers.

Objectives can be illustrated as follows.

1. Critically analyze and understand the problem
2. Identify functional and non-functional requirements
3. Design an overall architecture
4. Identify required technologies to address the issue
5. Study the technologies
6. Design and implement the system
7. Make recommendations to improve it further

### **1.6 Solution approach**

The solution is focused on creating a tool, which will assist developers in their development effort. This tool provides guidance by suggesting the most accurate programming solutions or technical documentation available to the developer. Though the proposed software system is basically focused on experienced software developers, it also facilitates for junior developers as a learning tool. Further the modern complex software systems require these sorts of plugins for monitoring code optimizations and testing.

Proposed solution aim at resolving few main issues when building a developer assistant tool as discussed below:

1. The tool should generate a query based on the in-progress code of the developer, which can be used for web/API search.
2. It should only obtain information from relevant sources therefore it has make use of well-known Q&A sites and documentation specific to programming language.
3. It should evaluate the search results using its content and meta data to further rank and present help accordingly.
4. This tool should also consider performance issues in the IDE. It should not delay operations or add overhead to IDE.

To achieve above-mentioned objectives, system is developed using separate modules. The system will consist of a query building module, search module and a ranking module. It is

essential to test this tool extensively to enhance its algorithms. With each test cycle, these three modules can be enhanced further.

### **1.7 Structure of the Thesis**

Next few chapters are aiming at giving a thorough knowledge about the suggested solution. Structure of the following chapters is as follows: Chapter 2 discusses the information, which have been collected from various studies that others have carried out to solve similar problems. Chapter 3 will give an introduction about the technology adapted. Next, Chapter 4 discusses the approach to solve the problem. Chapters 5, 6 and 7 describe the high-level design approach of the solution, implementation and discussion in order. In Chapter 8 it discusses the conclusion of the thesis and further work.

### **1.8 Summary**

In this chapter, it has discussed in detail about background and motivation, hypothesis, aim and objectives and approach of the solution. In the next few chapters it describes some key points about the system.

# Chapter 2

## Review of Others Work

### 2.1 Introduction

This chapter is about related research carried out by various scholars on this subject matter. Some of the studies cited below analyze the necessity of a developer assistant tool while other studies carried out implementations but focusing different aspects of such a tool. The following research survey and the table of comparisons describe those similar research areas and studies that already implemented such applications.

### 2.2 An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks

The goal of this research [5] was to investigate developers' strategies for understanding and utilizing relevant information and discover ways in which Eclipse and other environments might be related to strategies. Researcher tried to take a closer look at variations in developers' task structures, task contexts, and perceptions of relevance. The research investigation only considered one programming language and one development environment.

The limited size of sample and the limited experience of the developers in the sample both limit the study's generalizability. Research describes program understanding as a process of searching, relating, and collecting relevant information; all by forming perceptions of relevance from cues in the programming environment. This model is an extension of the general theory of information foraging [6]. This model is consistent with prior models of program understanding, while accounting for developers' actions.

### 2.3 COSME: A NetBeans IDE plugin as a team-centric alternative for search driven software development

The goal of this study [7] was to develop an IDE plugin for NetBeans IDE, which helps developers with search-driven development. Although search-driven development is



considered an individual task, this research is focused on how information retrieval applications can be more collaborative. So the idea behind this study is to provide a tool that developers can use for collaborative information retrieval. COSME (COde Search Meeting) tool can be used by remotely located developers to share search sessions or any other related documentation or information among them. COSME includes features such as automatic division of labor, search sessions management and explicit recommendations.

This tool is more focused as a distributed collaborative environment than trying to analyze the current code developer is working on. This tool is more suitable for larger software development companies where many developers are dealing with same type of technologies and source code. They will have more common issues that they need to look for. In that case, junior developers might be able to use more refined search carried out by more experienced developers.

#### **2.4 SurfClipse: Context-Aware Meta Search in the IDE**

Researchers of SurfClipse [8] are focused on mitigating time spent on fixing exceptions that occur during software development. Programming exceptions can have different levels of interpretation having an exception stack rather than a simple error message and it is not a simple task for a developer to manually analyze the error messages given the IDE to determine the issue. Developers normally would perform time-consuming, tedious debugging steps to analyze the exception even before looking in the web for a solution. So, it is important to understand the context of the code before querying the web for a solution for the exception.

SurfClipse is developed as an IDE assistant tool that analyzes an encountered exception and its context in the IDE, and recommends suitable search results via web searches right inside the IDE or provides search queries that can be used in the web. SurfClipse works as both a proactive and interactive tool.

The researchers proposed a tool that captures technical details of an encountered exception, and generates a list of suitable search queries, which are ranked, based on the context of the

exception. These search queries can be used in the Surfclipse IDE tool itself or in the web. Then the search results are also ranked per the encountered exception and the context of the code.

Web search is performed using three popular search engines and StackOverflow site via API calls and make use of StackOverflow metrics such as most recent and relevant post for ranking

This research is more focused on analyzing an encountered exception at the time of debugging/testing a program rather than at the time of development

### **2.5 Prompter: A Self-confident Recommender System**

Prompter [9] is another tool developed which is quite similar to Surfclipse that automatically search relevant discussions from StackOverflow in order to assist developer. In this research, it looks for in-progress code context rather than encountered exceptions to generate the search query.

Prompter is a plug-in for the Eclipse IDE. It identifies relevant StackOverflow discussions, evaluates their relevance given the code context in the IDE, and notifies the developer if and only if a user-defined confidence threshold is surpassed. Prompter consist of Query generation service, Search service and ranking model.

One of the novel approaches in this research is generating queries from code context. Researchers made use of current natural language processing mechanisms to generate search query. They also computed the entropy of all terms present in Stack Overflow discussions by using the data dump of June 2013, which was used for query generation. Entropy and term frequency are used as two factors eliminating certain words or to give weightage to words. Thus, it is very sensitive to the data dump used to calculate entropy.

## 2.6 Comparison of other approaches to DevAssist.

	<b>Other approaches</b>	<b>DevAssist</b>
<b>SurfClipse: Context-Aware Meta Search in the IDE</b>	<p>Analyzes an encountered exception and its context in the IDE, and recommends not only suitable search queries but also relevant web pages for the exception (and its context).</p> <p>Eclipse plugin</p>	<p>Analyze the in-progress programming code rather than the exception of the program.</p> <p>Visual Studio plugin.</p>
<b>Prompter: A Self-confident Recommender System</b>	<p>Proposed a novel approach that, given a context in the IDE, automatically retrieves pertinent discussions from Stack Overflow, evaluates their relevance, and, if a given confidence threshold is surpassed, notifies the developer about the available help. [ref]</p> <p>Uses entropy and term frequency for query generations.</p> <p>Eclipse plugin</p>	<p>DevAssist uses MSDN, Stack Overflow as well as other web searches to provide recommended solutions.</p> <p>In addition, DevAssist uses meta tags to analyze content of web results and retrieve code snippet and rank them using Levenshtein distance between terms occurred in title of the results rather than using entropy.</p> <p>DevAssist is developed as a Visual Studio Plugin.</p>

Table 2.1: Comparison between other's work and our solution

## **2.6 Summary**

In this chapter, we clarified similar studies others have carried out and how the knowledge about those studies were based for this project. Next few chapters discuss more in depth representation of this project including the technology, design and implementation of DevAssist.

## Usage of Technology

### 3.1 Introduction

In this chapter, it is discussed about technologies it had used to solve the problem. Focus of this chapter is to state how those technologies can be used efficiently to design a solution for the problem. There are special reasons to use each of these technologies within this approach to perform specific tasks.

### 3.2 Technologies Used

Technologies are crucial factor when implementing a system. Developers should pay a lot attention on its suitability and the compatibility when developing the system.

Here are the reasons why it is needed to use correct technologies when developing the system,

1. When developed, some technologies may provide high performance compared to others.
2. For a given system some technologies are easy to use and implement.
3. Some technologies are specifically designed for specific type of systems.
4. Some technologies provide greater support in developing compared to other technologies.
5. New and enhanced technologies provide greater user friendliness compared to other technologies.
6. Environment which they are going to be used will be critical in deciding the suitable technology.

Therefore, by taking the above-mentioned factors into consideration, a thorough study must be done to find suitable technologies for the solution. By analyzing deep accordance, the

matter, the Microsoft technologies are selected as best matching technologies to develop the solution.

### **3.3 Microsoft technologies**

Microsoft was founded by Paul Allen and Bill Gates on April 4, 1975, to develop and sell BASIC interpreters for the Altair 8800. It rose to dominate the personal computer operating system market with MS-DOS in the mid-1980s, followed by Microsoft Windows.

#### **3.3.1 What are Microsoft Technologies?**

There is bulk of technologies launched by Microsoft since it was begun. File systems, backup, administration, internet services, databases, developer services are some of among them.

#### **3.3.2 .NET Framework**

Microsoft started development on the .NET Framework [10] in the late 1990s originally under the name of Next Generation Windows Services (NGWS). By late 2001 the first beta versions of .NET 1.0 were released. The first version of .NET Framework was released on 13 February 2002, bringing managed code to Windows NT 4.0, 98, 2000, ME and XP.

#### **3.3.3 What are the components of .Net Framework?**

.Net Framework is consisted with two components basically which are Common Language Runtime (CLR) and .Net Framework Class Library (FCL) [10].

Common Language Runtime (CLR) provides an environment to run all the .Net Programs. The code which runs under the CLR is called as Managed Code. Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management and thread management.

This is also called as Base Class Library and it is common for all types of applications i.e. the way you access the Library Classes and Methods in VB.NET will be the same in C#,

and it is common for all other languages in .NET. The following are different types of applications that can make use of .net class library.

1. Windows Application.
2. Console Application
3. Web Application.
4. XML Web Services.
5. Windows Services.

### **3.3.4 Why .Net Framework?**

The .Net Framework offers several advantages to developers. Below are few advantages of .Net Framework

1. Consistent programming model
2. Direct Support for Security
3. Simplified Development efforts
4. Easy application deployment and Maintenance
5. Assemblies

### **3.4 Levenshtein distance**

The Levenshtein algorithm [11] (also called Edit-Distance) calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is by the dynamic programming approach. A matrix is initialized measuring in the (m, n)-cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the other word. The matrix can be filled from the upper left to the lower right corner. Each jump horizontally or vertically corresponds to an insert or a delete, respectively. The cost is normally set to 1 for each of the operations. The diagonal jump can cost either one, if the two characters in the row and column do not match or 0, if they do. Each cell always minimizes the cost locally.

### **3.5 Summary**

In this chapter, we discussed about technologies that we have used and going to be used to develop the web application. In the next few chapters let us look at how those technologies are adapted effectively to meet our requirements.



# Chapter 4

## **An Approach to DevAssist: Developer assistant for VisualStudio IDE**

### **4.1 Introduction**

Software developers often require knowledge beyond the one they possess. Here we describe a novel approach DevAssist: professional assistant for IDE based software development, DevAssist will analyze the code context to provide related discussions available in the top technological websites such as StackOverflow. These discussions will be available inside the IDE enabling the developers to have code suggestions right at their fingertips. The approach is further elaborated in the sections: hypothesis, input to the system, output of the system, process, users of system and features of the system.

### **4.2 Hypothesis**

The hypothesis of this research is to propose an algorithm to derive the context of the code and pass it as a query to API services of top technology related websites to give suggestions. The returned results from these websites are then prioritized based on different factors and propose solutions/suggestions to developers to enhance their coding.

### **4.3 Input to the system**

In progress code snippets that developer is working on can be an input to the solution. That can be a fully or partially completed method, class/method declaration or even a library related object creation. Additionally, system provides manual search capability discussion through DevAssist plugin. So, the manually query's also taken as input to the system. From beginner to expert developers can benefited by the system by taking their level of sensitivity to the related search.

### **4.4 Output of the system**

The output of the system will be the most relevant discussion that can be benefited to the developer.

## **4.5 Users of the system**

Different users can benefit from novel DevAssist system in multiple ways. More importantly, professionals, academics, and students working within the systems development life cycle who care about creating, delivering, and maintaining software can be directly benefited by this solution.

## **4.6 Process**

There are three main processes in entire construction of this hypothesis, namely, query building, searching and ranking. These three principles are very different and they are studies of their own. In the following paragraph, brief introduction to each process is given, and then continue to investigate each process in detail.

Main purpose of query building process is to make a valid query from automatically captured code context in the IDE. Secondly, searching process invokes relevant API calls on technical websites and direct searching on different search engines. Ranking model will prioritize the right discussions to suggest to developer.

### **4.6.1 Query Building Process**

To get the most suited discussion for a given code, it is a must to have a good strategy in building an accurate query to trigger the searching.

This process is the most challenging part of the whole project thus generating successful query will help other processes work seamlessly as well.

#### **4.6.1.2 Generate query from the code context**

Query generator module uses a naive approach to treat the code as a bag of words by splitting and removing unnecessary comments, variable declarations and stop-words [12] [13]. Then the remaining code is tokenized with the help of natural language toolkit [NLTK] [14]. Separation of programming language oriented pre-defined keywords and pre-defined contextual keywords from the tokens happens next. Remains are prioritized per the frequent word count. Top most frequent terms are selected and query composed out of those tokens appropriately describe the code context.

## **4.6.2 Searching process**

After building the query with the addition header data it is sent to the search service. In general, there are two kinds of searching strategies. Under the first searching strategy, APIs of well-known Q&A sites are invoked with the encoded query. One of the disadvantage of this approach is it is not guaranteed that there will be a matching the query with available discussions. In the second searching strategy, query is sent to the search engines to performed web search. Therefore, second searching strategy will be proposed to solve the disadvantages of the first strategy.

After taking the resulting URLs duplicates are removed. Each result coming from API calls and Web search are sent to the Ranking Model for further processing.

### **4.6.2.1 API Service**

API service responsible to handle multiple API's and calls, API is an application programming interface for either a web server or a web browser, we can directly call configured web APIs with query generated previously. Before we use the service, it should be registered on particular website to get a request key. Request keys grant requests per day, and are necessary to use access tokens created via authentication. All responses are JSON, Every response in the API is returned in a common "wrapper" object, for easier and more consistent parsing. Information's coming from the API clean and sent to the ranking model to further processing. Well known Q&A sites as well as Google provides API service to search internally that works with .NET framework. However, Google search API is a paid service and StackOverflow API [15] [16] has a limitation per day.

Example service call:

```
https://api.stackexchange.com/2.2/search/advanced?order=desc&sort=activity&title=zip%20file&site=stackoverflow
```

### **4.6.2.2 Web Search**

First part of web search is clutching relevant URLs. Query is sent to search engines like (Google, Bing, Yahoo) to perfume web search, first 20 URLs of each search engines are collected. Every URL that refers to discussion must match with certain pattern. Others are discarded.

Example:

Stack Overflow: `http://stackoverflow.com/questions/<id>/<title>`

Otherwise it is discarded.

Example web search:

`https://msdn.microsoft.com/en-us/library/ms404280(v=vs.110).aspx`

`stackoverflow.com/questions/940582/how-do-i-zip-a-file-in-c-using-no-3rd-party-apis`

Collected URLs are merged and duplicates are removed. Second part of web search is to visit each page to look necessary information like, Meta information's, code segments. If code segment is not found, those URLs are also discarded. Finally, collected information is saved under Temp folder for further reference.

### **4.6.3 Ranking model**

#### **4.6.3.1 API call Ranking model**

Goal of the API ranking model is to rank the retrieved discussions and measure their relevance to the query by considering some features mentioned below. Considering those features system must assign a value to retrieved discussions. Ranking relies on different features like textual similarity, tag similarity, user reputation, view count, accepted answer score and the question score those are briefly explained below. The ranked list of discussions is sent to the plug-in interface to notify Developer.

### 1. Textual similarity

This checks similarity of the code on the IDE to the textual part of the discussions. E.g.: topic of the question, body and coding part. By using stop word removing and duplicate removing part of building query we have created array of words from textual part of discussion to compare with the query build before.

### 2. Tag similarity

Tags of the code context are matched against the obtained discussion tags. tags of code context split to remove versions and symbols (e.g.: visual-c#2015-Compiler-version-4.0.30319.1, C#-5- .net4.5 become c#, .net4.5). Tags from discussion retrieves are listed on the JSON data what we retrieved from the API service ("tags": ["c#","2015",".net"]).

### 3. User reputation

The level of reputation of the person who posted the question by considering that evaluates the reliability of the person who asked the question on the community.

### 4. Accepted answer score

This is about the quality of the score of the accepted answers in the discussion. In case accepted answer not present those are not taken in to account. It can be even detected by using attribute of "is\_answered: true/false" property of the returning JSON.

### 5. Question score

This indicates the quality of the question according to the community.

#### **4.6.3.1 Web search data ranking**

Collected Meta information from search phase is used for prioritization in the ranking algorithm. Goal of web search data raking process is to get the highest priority URL first and display others after that to developer. According to the ranking process most relevant

URL get more score. Meta information like page titles is tokenized into words. Each token from every title is compared with rest of the page titles to find a matching token. For that we use Levenshtein algorithm. Levenshtein algorithm is word comparison algorithm, which returns a value per relevance (distance) of each character to character from comparing word. It helps to compare collected Meta title to one to another.

#### **4.7 Features**

Following features are assuring by the DevAssist plugin.

1. Robustness

Ability of a DevAssist plugin to cope with errors during execution and cope with erroneous input.

2. Focus on questions about actual problem/difficulties you have faced.

System takes on time code context as input to the system.

3. Best answers show up first so that they are always easy to find.

Most up voted and relevant answers always show up first

4. Real time retrieval most updated answers

Working real-time with most accurate and up-to-date sources of information available

5. Minimal user input

Other than the code context what he or she working only sensitivity adjust bar is the input to the system.

#### **4.8 Summary**

This chapter describes overall solution of the research. Here we have mention input to the system, output, each process with sub processes, users and features of the system. Next chapter describes design of our solution.

# Chapter 5

## Analysis and Design

### 5.1 Introduction

This chapter describes the analysis and design of the system with more details. Here it has explained the design diagram of our project and the task of each module in the system and interaction among those components. This chapter will help to get a clear view of the design.

### 5.2 The Proposed Design

The ultimate outcome of the system is, provide or suggest accurate results for given problems of software developers, which will arise in their ongoing software developments. For this purpose, the system must have a clear understanding about the source code and programming language. Query data is captured in two different methods as explained below.

- 1) Module that captures data manually, entered by the user.
- 2) Module which gathers data automatically from the system.

#### 5.2.1 Module which captures data manually entered by the user

The manual data gathering is planned to do by providing a text box controller in the user interface. Figure 5.1 illustrates the functionality of this module.

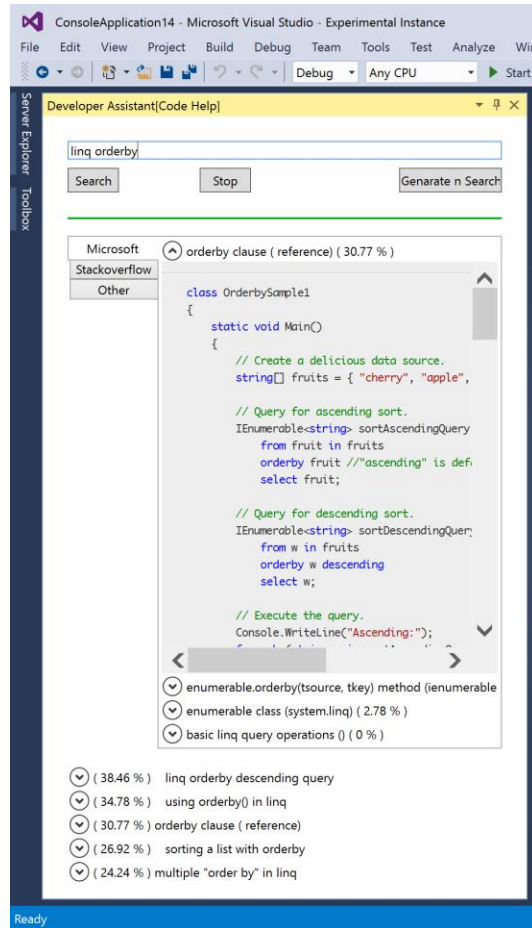


Figure 5.1: Module that captures data manually from the user

- Search text box

It captures search queries that developer manually typed in. This task is totally done by the user when he or she needs a help from the plugin. The target user of the system is a software developer. Therefore, developer can solve the problem in the same environment with developments without running any browser applications separately.



## 5.2.2. Module which gathers data from code editor

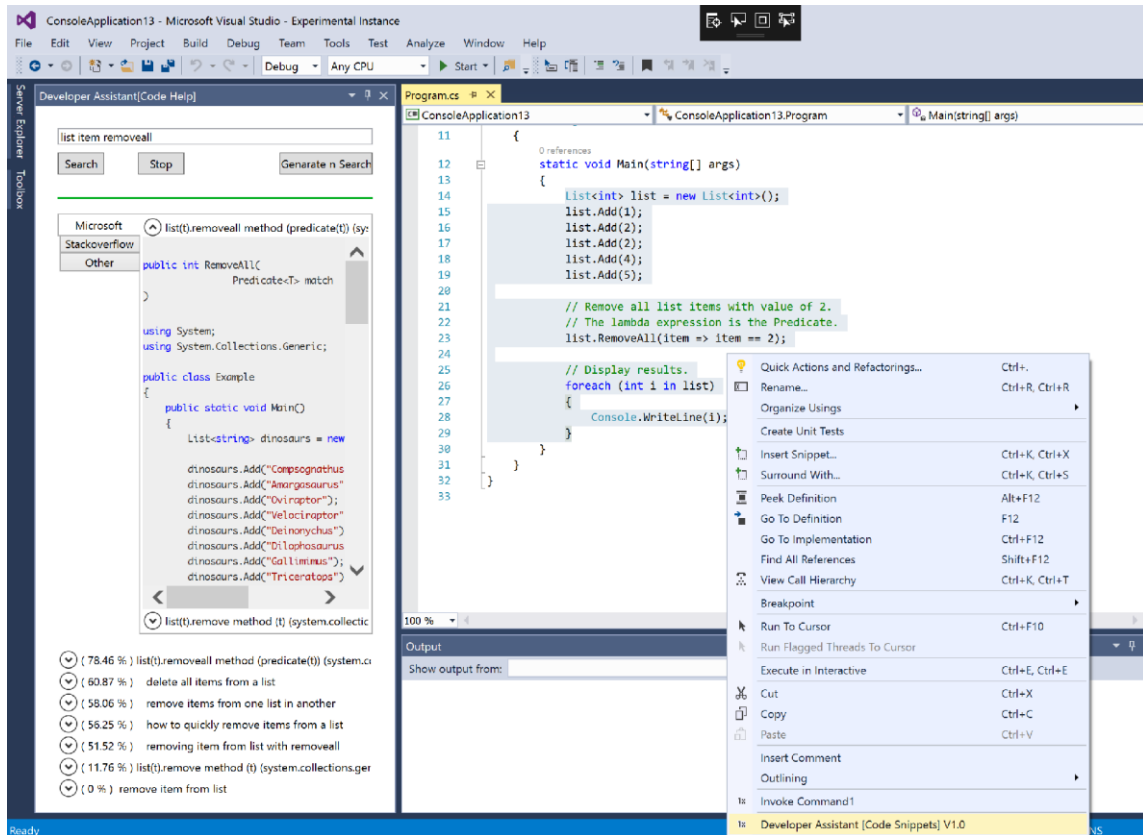


Figure 5.2: Module which gathers data from code editor

This task is partially done by the user and the rest of phase done by the system automatically.

DevAssist provides user with a few different ways to select code segments and ask for help or right click the exception to get help and even manually type query to seek help. The process is as follows:

1. Remove single line and multiple line comments.
2. Separate variable declarations.
3. Tokenize the remaining codes.
4. Weighting and find the frequency of the keyword and contextual keyword.
5. Find the distance between each word to calculate similarity.

6. Choose the top scored five tokens

### 5.3 Design Diagram

The designed diagram of the implemented system with its components is shown below.

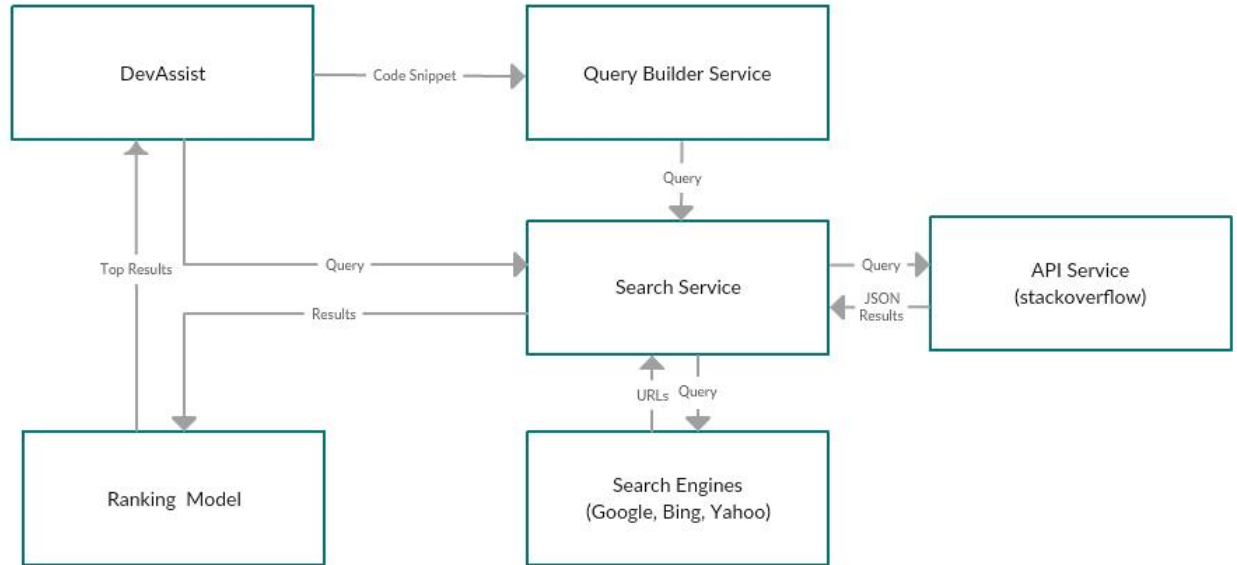


Figure 5.3: Design diagram of the system

Figure 5.3 illustrates the top-level architecture of DevAssist, with following components

1. Interface Module
2. Query Build Module
3. Search Module.
4. API service Manager
5. Ranking module.

### 5.4 Detailed description about the Design Diagram

Detailed description about each module of the design diagram could be expressed as below.

- Query Builder Service.

The plug-in is designed to generate a query automatically. Very first approach of our work is to get a highest priority word from selected code snippet. The approach of building the query is to treat the code as a bag of words by: (i) splitting identifiers and removing stop

words; (ii) ranking the obtained terms per their frequency; and (iii) selecting the top-n most frequent terms. Simply removing stop words and ranking per frequent words will not provide the most accurate highest priority item. As natural languages (NLTK), source codes don't contain too much of stop words, other than variable declarations. Therefore, following few steps like removing unnecessary comments, assigning different weightage to each code level (keyword, contextual keywords, declared variables, remaining words), the list of highest priority words will get selected to be used for query generation.

- Search Service

Search service acts as a proxy within DevAssist plugin. Query generated from query service and the data entered manually are used as inputs to the search service. This process mainly consists with two parts which are search engine browsing and individual page visits. Other than the web browsing, the search service responsible for storing collected information on text documents for future reference. Those are stored in a temporary directory. In the next search step, those files are deleted.

- Ranking Model

The goal of the ranking model is to prioritize retrieved discussions per the relevancy with the search query. Each title information is compared with the rest of the titles from other sites to check the similarity. It relies on different features that capture relations between retrieved discussions and source codes.

Semantic similarity ratio with Levenshtein Distance: The goal is to assess the similarity between each topics / titles of the pages and the meta information of the discussion page. For this we must first remove the unwanted html tags and special characters (like ASCII) then remove the English stop words. Remaining words are going to process with Levenshtein Distance checking algorithm. Levenshtein Distance is a metric for measuring the amount of difference between two sequences. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other.

- Search Engine

Query from query build service and manually entered text used to obtain results from search engines. The query is send to world top rated search engines like Google, Yahoo, and Bing by Microsoft for web search. The first 20 of each search engine results are collected. StackOverflow and Microsoft developer network (MSDN) web sites follow special format to catch up only discussion threads.

For example: <https://www.stackoverflow.com/questions/{id}>,  
<https://msdn.microsoft.com/en-us/library>

All the results are collected and merged to remove duplicates. At that time, those links are neither visited nor ranked.

- Page Visit

Collected URLs of each website are processed in background by visiting each link from search service to collect title, Meta information and code segments. Code segments are filtered by using W3C certified HTML5 tags.

- `<code></code>`
- `<samp></samp>`
- `<pre></pre>`
- `<blockquote>`

Those tags are generally contained in the context (i.e., stack traces and code segments) associated with the discussed exception in the page. Code segment availability can be checked using this process. Collected information is stored in a text file. According to the unique number that has been given to the text files, the pages won't be revisited to show the final output.

Title and Meta information of each page send to ranking model to rank the result list.

## 5.5 Sequence Diagram of the System

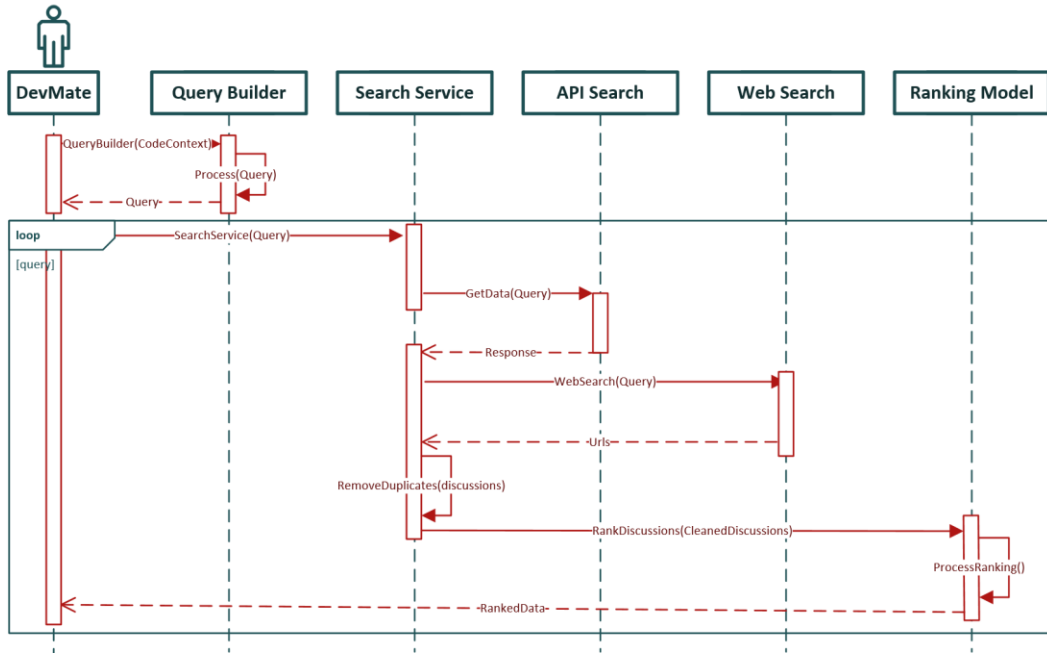


Figure 5.4: Sequence diagram of the system

## 5.6 Summary

The design has been described with focusing on few major modules and what each module does. Those modules are namely query generation module, searching and ranking. This system is designed to leverage Visual Studio in built functionalities. The illustrated design has been used for the implementation, which is described in next chapter.

## Implementation

### 6.1 Introduction

This chapter describes implementation related aspects of this project. General design decisions have been described in the previous chapter, including the system and software design concept. It provides the technical information about the overall solution, including the system and software design decisions taken and the implementation of query builder service.

### 6.2 Overall solution

The implementation of the DevAssist is written as a plug-in for Visual Studio using the plug-in development environment in Visual Studio IDE (Integrated Development Environment). It is C# language based, platform independent solution which can be installed with the 2013 versions of Visual Studio or above. The plug-in is available for free downloading via Visual Studio extension/package manager. It does not require any other pre-requisites other than ongoing internet connection and Visual Studio IDE.

### 6.3 Implementation of Query Builder Service

Query builder is responsible for building a query to retrieve relevant discussions on selected web sites. The DevAssist shortcut is placed as a sub menu in the 'Edit' menu and in the default right click menu within the editor.

Stepwise, query builder processes as follows:

1. Get the selected code snippet from the code editor by highlighting.
2. Right-click on the selected code snippet and choose "Support from DevAssist".

Figure 6.1 illustrates how to use DevAssist.

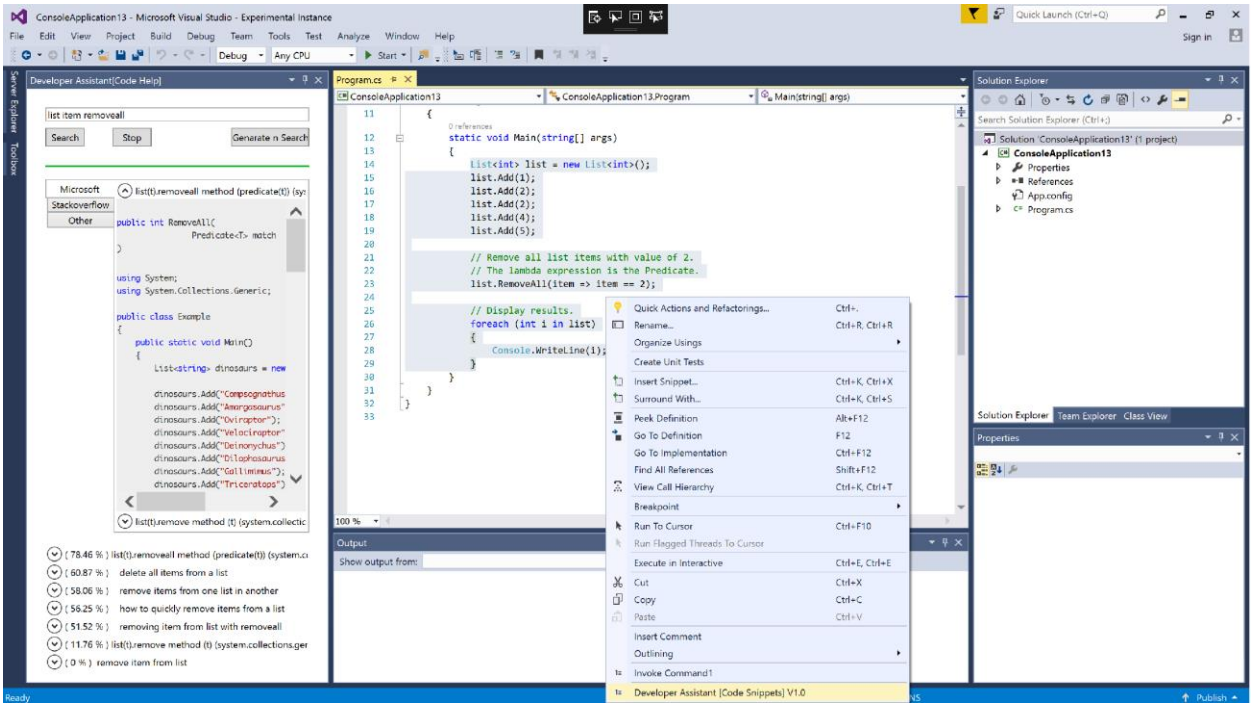


Figure 6.1: Screenshot of Visual Studio IDE with DevAssist

The selected code segment proceeds in several ways as mention in design chapter.

### Step 1. Remove single line and multiple line comments.

The purpose of removing comments is to get a plain code. The below regular expressions are used for this task.

```
var blockComments = @"\/\*(.?)\*/";
var lineComments = @"//(.?)\r?\n";
var strings = @"\"\"((\\[^\n]|^[^\n])*)\"\"";
var verbatimStrings = @"@(\"\"[^\"]*\"\")+";
var consoleWriteLine_ = @"Console.(.?)\\((.?)\\)";
var betweenQuotes = @"\".??\"";
```

### Step 2. Separate variable declarations.

The below regular expressions catch the most of variable declaration patterns inside c# code. The separated unique variable declarations are weighted and prioritized according to the frequency.

```
[A-Za-z0-9]+(\s[A-Za-z0-9_]+)?;";
[A-Za-z0-9]+(\s[A-Za-z0-9_]+)+(\s[=]+)(\s[A-Za-z0-9._*]+)?;";
```

### Step 3. Tokenize the remaining codes.

Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing. Below code snippet explain the implementation of tokenization.

```
public List<string> Tokenize(string text)
{
    string[] tokens = text.Split(delimiters_no_digits,
StringSplitOptions.RemoveEmptyEntries);
    tokens = tokens.ToList().Where(p => p.Length >
1).ToArray();//length more then 1

    for (int i = 0; i < tokens.Length; i++)
    {
        string token = tokens[i];

        // Change token only when it starts and/or ends with ""
and
        // it has at least 2 characters.
        token = token.Replace(System.Environment.NewLine,
string.Empty).Replace("@", string.Empty);
        if (token.Length > 1)
        {
            if (token.StartsWith("") && token.EndsWith(""))
                tokens[i] = token.Substring(1, token.Length - 2);
// remove the starting and ending ""

            else if (token.StartsWith(""))
                tokens[i] = token.Substring(1); // remove the
starting ""

            else if (token.EndsWith(""))
                tokens[i] = token.Substring(0, token.Length - 1);
// remove the last ""
```



```

        }
    }

    return tokens.ToList();
}

```

#### Step 4. Segregate keyword and contextual keyword.

There are some common keywords and contextual keywords in C# [17] and VB. Net. Keywords are predefined, reserved identifiers that have special meanings to the compiler. They cannot be used as identifiers in your program. Few example of keyword shows below and rest of attached to appendix.

abstract	as	base	bool
break	byte	case	catch
char	checked	class	etc

Table 6.1: C# keywords

A contextual keyword is used to provide a specific meaning in the code, but it is not a reserved word in C#. Some contextual keywords, such as partial and where, have special meanings in two or more contexts.

add	alias	ascending
async	await	descending
dynamic	from	get
yield		etc

Table 6.2: Contextual keywords in C#

### Step 5. Prioritize token categories and calculate term frequency

Once the keywords and contextual keywords are segregated they are given lower priority than the remaining words.

Tokens are prioritized in below order:

- 1) Remaining words
- 2) Contextual Keywords
- 3) Keywords

Afterwards the term frequency of each token is calculated. Below code snippet illustrates the algorithm:

```
public Dictionary<string, int> ToStrIntDict(string[] words)
{
    Dictionary<string, int> dict = new Dictionary<string, int>();

    foreach (string word in words)
    {
        // if the word is in the dictionary, increment its freq.
        if (dict.ContainsKey(word))
        {
            dict[word]++;
        }
    }
}
```

```

        }
        // if not, add it to the dictionary and set its freq = 1
        else
        {
            dict.Add(word, 1);
        }
    }

    return dict;
}

public Dictionary<string, int> ListWordsByFreq(Dictionary<string, int>
strIntDict, SortOrder sortOrder)
{
    // Copy keys and values to two arrays
    string[] words = new string[strIntDict.Keys.Count];
    strIntDict.Keys.CopyTo(words, 0);

    int[] freqs = new int[strIntDict.Values.Count];
    strIntDict.Values.CopyTo(freqs, 0);

    //Sort by freqs: it sorts the freqs array, but it also
rearranges
    //the words array's elements accordingly (not sorting)
    Array.Sort(freqs, words);

    // If sort order is descending, reverse the sorted arrays.
    if (sortOrder == SortOrder.Descending)
    {
        //reverse both arrays
        Array.Reverse(freqs);
        Array.Reverse(words);
    }

    //Copy freqs and words to a new Dictionary<string, int>
    Dictionary<string, int> dictByFreq = new Dictionary<string,
int>();

    for (int i = 0; i < freqs.Length; i++)

```

```

    {
        dictByFreq.Add(words[i], freqs[i]);
    }

    return dictByFreq;
}

```

## Step 6. Generate query using term frequency and priority

Query should contain minimum three terms or maximum five terms. To select these terms, program initially consider remaining words bucket. If it has more than 3 unique terms, up to 5 terms are selected from this list. Otherwise, program will consider contextual keywords or keywords respectively.

Below code snippet explain the algorithm used to build the search query using maximum 5 terms:

```

//get first 5 items from dictionary to build the query
string buildedQuery = string.Empty;
if (sortedTFTokens != null)
{
    if (sortedTFTokens.Count > 4)
    {
        int count = 0;
        foreach (var item in sortedTFTokens)
        {
            if (count == 5)
                break;
            buildedQuery += item.Key + " ";
            count++;
        }
    }
    else
        foreach (var item in sortedTFTokens)
            buildedQuery += item.Key + " ";
}
if (buildedQuery != string.Empty || buildedQuery != " ")

```

```

        {
            searchKey = buildedQuery.Replace("\"",
string.Empty).Replace("\\", string.Empty);
            this.txtSearch.Text =
buildedQuery.Replace("\"", string.Empty).Replace("\\", string.Empty);
            Button_Click(new object(), new RoutedEventArgs());
        }

```

Below code segment illustrate selecting terms using different token sets per their priority.

```

//TERM FREQUENCY METHOD-----
-----

List<string> remaingTokens = contextualKeywordNotMatch;
List<string> _contextualKeywordList = contextualKeywordMatch;
List<string> _keywordList = keywordMatch;
List<string> _variables = variableNames.Split(',').ToList();
//contextual keyword TF
//keyword TF
//variable names TF
Dictionary<string, int> tfTokens;
Dictionary<string, int> sortedTFTokens = null;

if (remaingTokens.Count > 2)
{
    tfTokens = termFrequency.ToStrIntDict(remaingTokens.ToArray());

    sortedTFTokens = termFrequency.ListWordsByFreq(tfTokens, T
ermFrequency.SortOrder.Descending);
}
else
{
    if (_contextualKeywordList.Count > 2)
    {
        tfTokens = termFrequency.ToStrIntDict(_contextualKeywo
rdList.ToArray());

        if (remaingTokens.Count > 0 && !string.IsNullOrEmpty(r
emaingTokens[0]))

```

```

        tfTokens.Add(remainingTokens[0], 100); //add remaining
token and prioritize as 1st
        if (remainingTokens.Count > 1 && !string.IsNullOrEmpty(remainingTokens[1]))
            tfTokens.Add(remainingTokens[1], 99); //add remaining token and prioritize as 2nd
            sortedTfTokens = termFrequency.ListWordsByFreq(tfTokens, TermFrequency.SortOrder.Descending);
        }
        else
        {
            if (_keywordList.Count > 2)
            {
                tfTokens = termFrequency.ToStrIntDict(_keywordList.ToArray());
                if (remainingTokens.Count > 0 && !string.IsNullOrEmpty(remainingTokens[0]))
                    tfTokens.Add(remainingTokens[0], 100); //add remaining token and prioritize as 1st
                if (remainingTokens.Count > 1 && !string.IsNullOrEmpty(remainingTokens[1]))
                    tfTokens.Add(remainingTokens[1], 99); //add remaining token and prioritize as 2nd
                if (_contextualKeywordList.Count > 0 && !string.IsNullOrEmpty(_contextualKeywordList[0]))
                    tfTokens.Add(_contextualKeywordList[0], 98); //add contextual keyword and prioritize as 3rd
                if (_contextualKeywordList.Count > 1 && !string.IsNullOrEmpty(_contextualKeywordList[1]))
                    tfTokens.Add(_contextualKeywordList[1], 97); //add contextual keyword and prioritize as 4th
                sortedTfTokens = termFrequency.ListWordsByFreq(tfTokens, TermFrequency.SortOrder.Descending);
            }
        }
    }
}

```

Full source code of each process has been attached in appendix chapter.

## 6.4 Implementation of Search Service

This tool uses client server architecture. Given that a user might be interested in refining the auto-generated search query or in a more traditional way of search, the tool provides a keyword-based search. The search is complemented with search query suggestion through auto-completion. The tool collects results in a nonintrusive way (i.e., without freezing the IDE). User can perform any other task while searching is processed in a separate thread in the background. Searching takes more than 80% of total execution time.

Considering query that build from query building module search service call Microsoft web client silently to observe result from google, yahoo, Bing. Figure 1 shows the code snippet of web client implementation.

```
public List<TechUrl> SearchQuery(string browser, string query)
{
    List<TechUrl> __TechUrls = new List<TechUrl>();
    webclient = new WebClient();

    if (browser == "yahoo")
    {
        string htmlContent = webclient.DownloadString(yahoo +
System.Uri.EscapeDataString(query));
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    else if (browser == "bing")
    {
        string htmlContent = webclient.DownloadString(bing +
System.Uri.EscapeDataString(query));
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    else if (browser == "google")
    {
        string htmlContent = webclient.DownloadString(google +
query);
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    return __TechUrls;
}
```

```
}
```

Each search engine call runs on a separate thread. All the process completes once the longest process of them is completed. After getting the first 20 search engine result set of each call we filter only the Uri, for that we use below regular expressions. Sample output attached to appendix.

```
string patternTitle = "(?i)<title>(.*?)</title>";
string patternMetaTags = "(?i)<meta([^\>]+)>";
string patternCodeSnippetTags = "(?i)<pre>(.*?)</pre>";
string patternCodeSnippetTags2 = "(?i)<pre([^\>]+)>(.*?)</pre>";

private string FilterTitleAndMeta(string htmlContent, bool title = true)
{
    Console.WriteLine("-----");
    MatchCollection _titleMetaContents = Regex.Matches(htmlContent, title
? patternTitle : patternMetaTags, RegexOptions.Singleline);
    if (_titleMetaContents.Count > 0)
        return removeTags(wordReplace(_titleMetaContents[0].Value));
    return string.Empty;
}
```

Merging all together and removing duplicates is the next part. By doing that, the full list of distinct Urls can be obtained. Second part of search process is to call individual URL to collect necessary information from each website. figure 3 shows below the implementation of calling page url separately. Calling process happens on a multithreaded environment to reduce execution time. Most of the pages are *loading* correctly but there are certain pages that have a long *delay* before *page loads*. It also happens with every newly created page.

Loaded sites data are filtered according to several groups 1. Meta information 2. Title information 3. Code snippets. Figure 4 shows the implementation steps of each information processing. Meta title information and code information validity checked with regular



expressions. META and title information are stored for ranking, available code snippet are validated and stored in a txt document under root path.

Html 5 can be filtered with below W3c certified tags

- <CODE>
- <PRE>
- <SAMP>
- <CODE-BLOCK>

Collected code snippets are saving under uniquely numbered documents under root path. Once the ranking happens we can show the pages to developer directly without going back to internet again.

## **6.5 Implementation of Ranking Model**

The goal of the ranking model is to rank the retrieved discussions information and assign them a value based on priority.

Since the search results are mainly retrieved from well-known search engines, results are already ranked to some extent. However, after receiving search results, meta data of those results are retrieved. Title included in meta data is mainly used in the ranking module in DevAssist. Each search result is given a priority value. Levenshtein algorithm is used to calculate similarity ratio of titles of the search results set, which is then used to calculate priority value.

- Levenshtein distance

Levenshtein distance algorithm [11] explains number of edits needed to turn one string into another. This metric is used to measure similarity and match approximate strings with fuzzy logic.

Example of Levenshtein distance algorithm:



```

// Step 1
if (n == 0)
{
    return m;
}

if (m == 0)
{
    return n;
}

// Step 2
for (int i = 0; i <= n; d[i, 0] = i++)
{
}

for (int j = 0; j <= m; d[0, j] = j++)
{
}

// Step 3
for (int i = 1; i <= n; i++)
{
    //Step 4
    for (int j = 1; j <= m; j++)
    {
        // Step 5
        int cost = (t[j - 1] == s[i - 1]) ? 0 : 1;

        // Step 6
        d[i, j] = Math.Min(
            Math.Min(d[i - 1, j] + 1, d[i, j - 1] + 1),
            d[i - 1, j - 1] + cost);
    }
}

// Step 7
return d[n, m];
}

```

The result of the distance of search sentence is not sufficient to determine if a discussion is to be recommended or not. Therefore, Levenshtein algorithm is adapted to compare similar words. So initially the similarity of each word is calculated with the rest of the words and best matching word is selected initially. Then the distance is calculated for a pair of sentences. Once the distance is calculated, similarity ratio is calculated using below function:

$$\text{Similarity Ratio} = 1 - (\text{Levenshtein Distance} / \text{Length of String})$$

Again, if we refer our previous example, Levenshtein Distance (D) is common for set (S, T). D is divided by the maximum length of string S or T. So, the Similarity ratio (R) is calculated as below:

$$R(S,T) = 1 - D / \text{Max}(\text{Length}(S), \text{Length}(T))$$

After calculating ratio for each set, we get set of strings with highest value order. Then the sum of similarity ratio in a single string is compared to rest of the sums in other strings to get the maximum value. If  $P(S_x)$  is the sum of similarity ratios for String  $S_x$ , and there is n number of search results,  $P(S_x)$  is calculated as below:

$$P(S_x) = \sum_{i=0}^n R(S_x, T_i)$$

where  $x \neq i$

The string with the highest number is considered to be the best match for search query.

Implementation of similarity ratio calculation:

```
public double GetSimilarityRatio(String FullString1, String FullString2, out double
WordsRatio, out double RealWordsRatio)
{
    double theResult = 0;
    String[] Splitted1 = FullString1.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
```

```

        String[] Splitted2 = FullString2.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        if (Splitted1.Length < Splitted2.Length)
        {
            String[] Temp = Splitted2;
            Splitted2 = Splitted1;
            Splitted1 = Temp;
        }
        int[,] theScores = new int[Splitted1.Length, Splitted2.Length]; //Keep
the best scores for each word. 0 is the best, 1000 is the starting.
        int[] BestWord = new int[Splitted1.Length]; //Index to the best word of
Splitted2 for the Splitted1.

        for (int loop = 0; loop < Splitted1.Length; loop++)
        {
            for (int loop1 = 0; loop1 < Splitted2.Length; loop1++) theScores[loop, loop1] = 1000;
            BestWord[loop] = -1;
        }
        int WordsMatched = 0;
        for (int loop = 0; loop < Splitted1.Length; loop++)
        {
            String String1 = Splitted1[loop];
            for (int loop1 = 0; loop1 < Splitted2.Length; loop1++)
            {
                String String2 = Splitted2[loop1];
                int LevenshteinDistance = Compute(String1, String2);
                theScores[loop, loop1] = LevenshteinDistance;
                if (BestWord[loop] == -
1 || theScores[loop, BestWord[loop]] > LevenshteinDistance) BestWord[loop] = loop1
;
            }
        }

        for (int loop = 0; loop < Splitted1.Length; loop++)
        {
            if (theScores[loop, BestWord[loop]] == 1000) continue;
            for (int loop1 = loop + 1; loop1 < Splitted1.Length; loop1++)
            {

```



```

        }
        BestWord[loop] = CurrentBest;
    }

    loop = -1;
    break;//recalculate all
}
}
}
for (int loop = 0; loop < Splitted1.Length; loop++)
{
    if (theScores[loop, BestWord[loop]] == 1000) theResult += Splitted
1[loop].Length;//All words without a score for best word are max failures
    else
    {
        theResult += theScores[loop, BestWord[loop]];
        if (theScores[loop, BestWord[loop]] == 0) WordsMatched++;
    }
}
int theLength = (FullString1.Replace(" ", "").Length > FullString2.Rep
lace(" ", "").Length) ? FullString1.Replace(" ", "").Length : FullString2.Replace(
" ", "").Length;
if (theResult > theLength) theResult = theLength;
theResult = (1 - (theResult / theLength)) * 100;
WordsRatio = ((double)WordsMatched / (double)Splitted2.Length) * 100;
RealWordsRatio = ((double)WordsMatched / (double)Splitted1.Length) * 1
00;

return theResult;
}

```

## 6.6 Summary

In this chapter, it had described the implementation of the process. It provides a clear understanding about the approach.

## Evaluation

### 7.1 Introduction

In this chapter, we will report on how we evaluate the solution to see whether objectives have been achieved. In additionally, a summary about the entire report, how our solution may differ from others work and the further works are also included on here.

### 7.2 Evaluation of the prototype

After the implementation of prototype, the prototype is tested with Visual Studio IDE version 2013 (Integrated Development Environment) and same environment which was used to developments. Then then solution is installed to the real devices which are different versions higher than 2013 of Visual Studio and checked the prototype again. The prototype is being developed onwards by increasing the suitability of an enterprise level usage.

#### 7.2.1 Generate search query

Below figures provide some sample queries generated by selecting a code segment and search results using DevAssist.

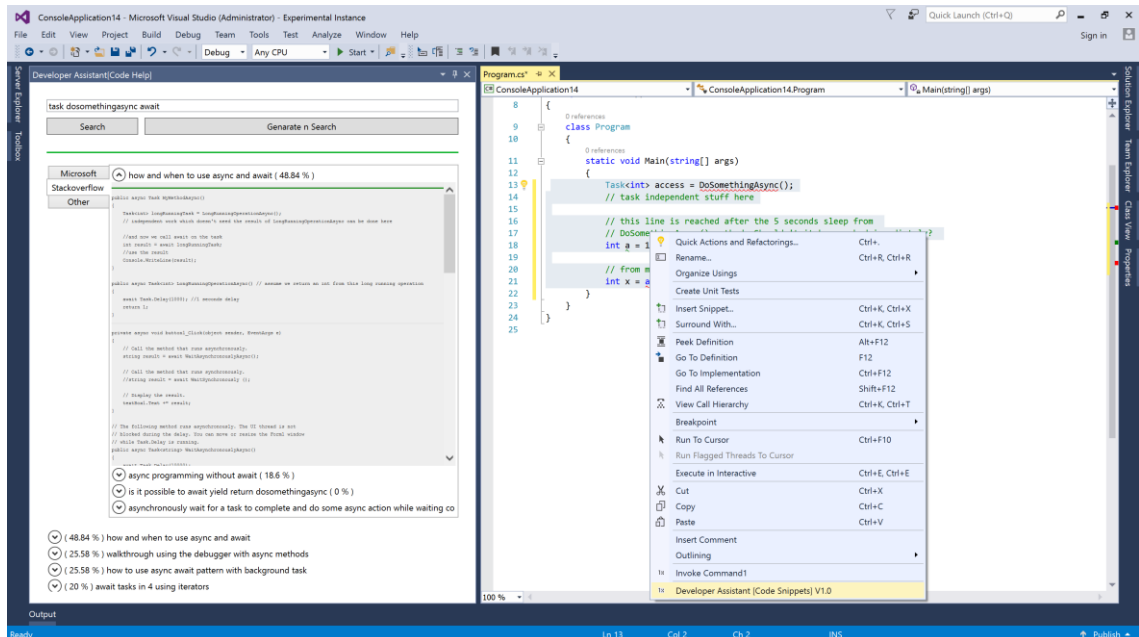


Figure 7.1: Query generation example 1



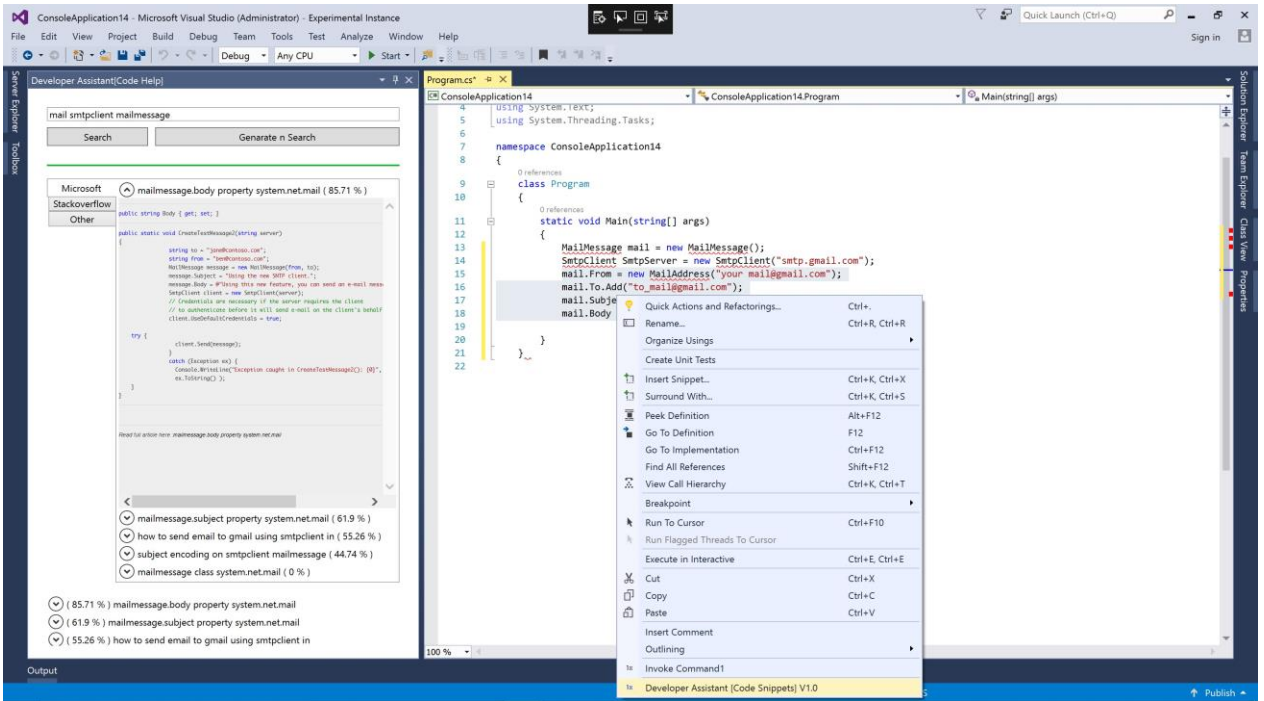


Figure 7.2: Query generation example 2

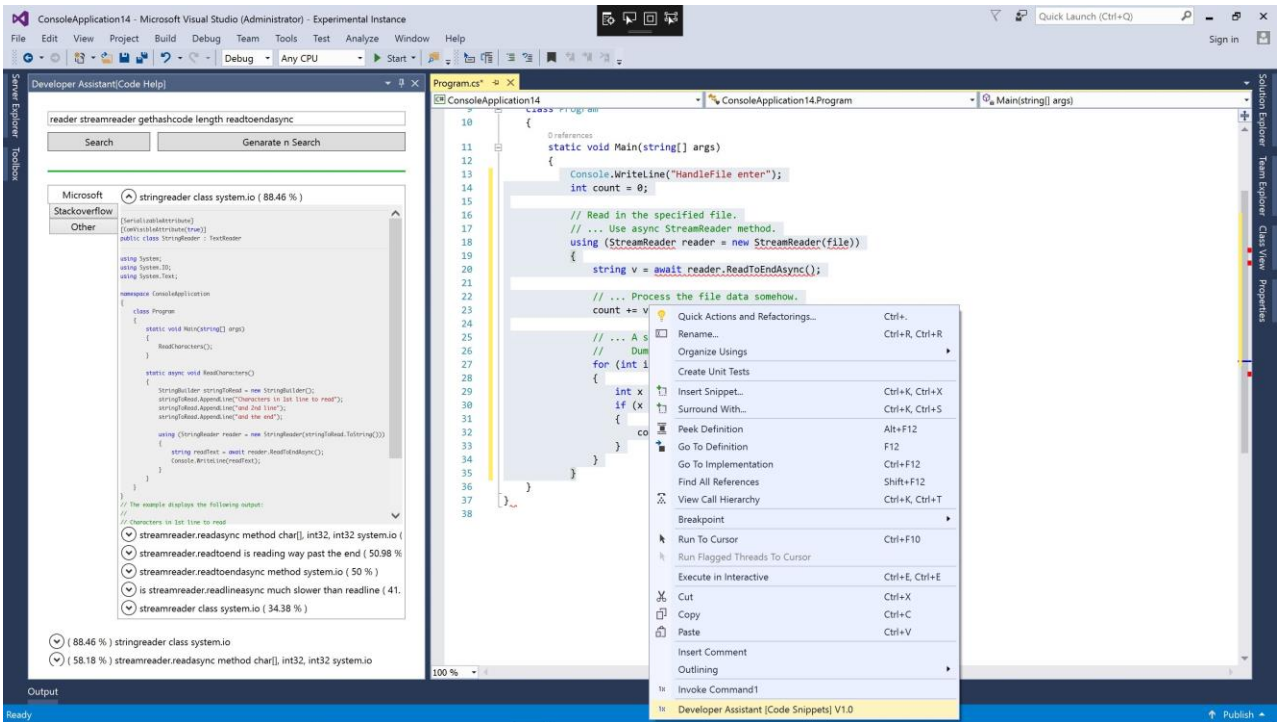


Figure 7.3: Query generation example 3

## 7.2.2. Ranking results

Below details illustrate some example of ranking of results for manually entered query strings.

- Query: stream reader c#

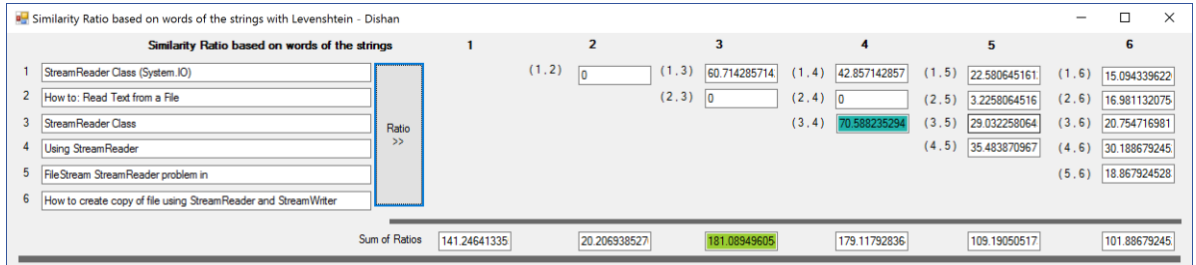


Figure 7.4: Ranking result 1

Result number 3 is the best answer. Order of the rest of the strings are 4,1,5,6,2.

- Query: async await programming c#

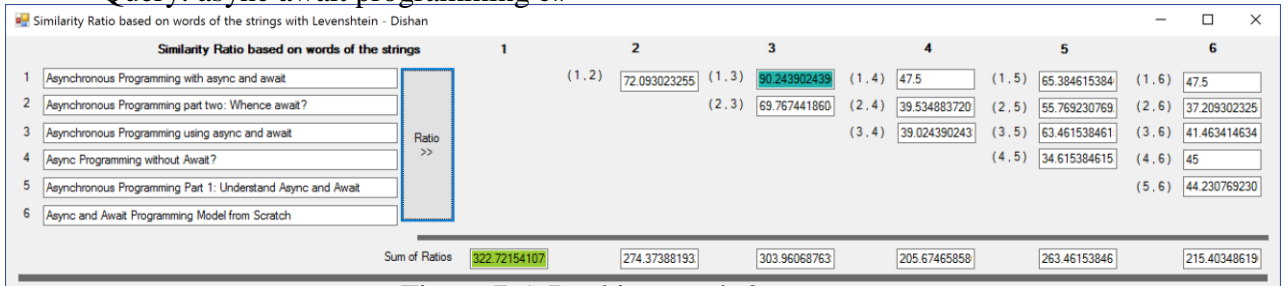


Figure 7.5: Ranking result 2

Result number 1 is the best answer. Order of the rest of the strings are 3,2,5,6,4.

- Query: random number generator c#

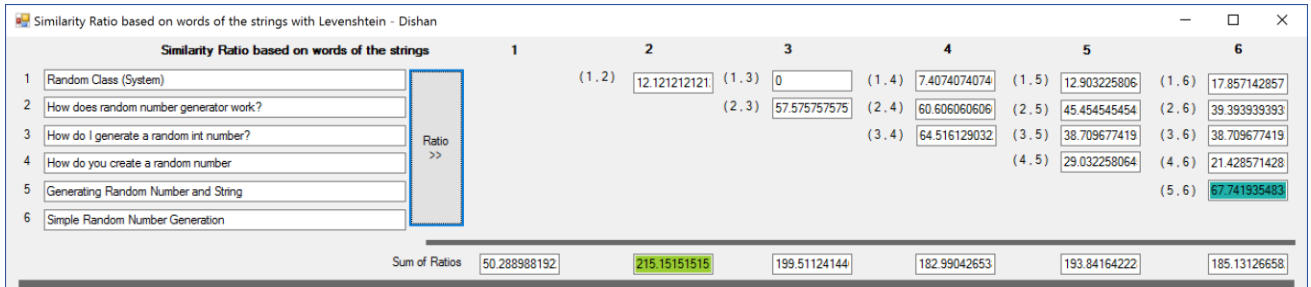


Figure 7.6: Ranking result 3

Result number 2 is the best result. Order of the rest of the strings are 4,1,5,6,2.

### 7.2.3 Accuracy of receiving a result using DevAssist plug-in

The user interface of the plug-in has a separate view screen which shows the accuracy of the returned results. Also, it shows the owner sites of the results. As an example, whether the results are received from StackOverflow site or MSDN etc. When considering a code segment analysis and generating query or manually typed search query, it will take maximum one minute to show the output result set when the ongoing Internet connection has no issues.

Pattern	Number of query taken to test	First displayed answer is the best answer	Generated Query describe the question properly	Resulted code snippet provides the answer	Average rating for results (out of 5)
Manually typed search query	30	62%	N/A	N/A	3.50
Auto built query	47	53%	43%	69%	3.23

Table 7.1: Accuracy of received results using DevAssist plug-in

Table 7.1 provides a summary of data collected using survey carried out with a group of 6 developers. They have been using DevAssist for their development tasks and completed questionnaire and provided a data set to analyze the tool. Manually typed in queries has better ratings according to this test results. The developers who tested this tool has a minimum of 3 years experience in the software industry. As these developers are experienced developers manual search queries they entered are very precise, therefore it resulted in better ratings. However, the results of the auto built query provides a better code snippet samples which has approval rating of 69%. Generate query has a poor rating of 43%. So these results show us that the concept of this project has good approval rating as the idea was to provide a better sample code snippet as outcome. Generated query might

not represent the day today search terms resulting in poor rating by viewers but it represents the coding task which will give a better outcome.

## 7.2 How our solution differs from the others' work

	Related Work	DevAssist plug-in
Use many web resources other than Stackoverflow	X	✓
Only manual typed search query	✓	X
Customize web resources	X	✓
No prerequisites	X	✓
No difficult configurations	X	✓
Independent from other API s	X	✓

Table 7.2: Evaluation of DevAssist against Related wok

There are some similar works to this solution that others have done before. Three of those are mentioned in section of related work. All of them are individually providing some kind of a solution for the problem that the system has required. The research work is based auto built query search resulting. It discusses about detecting ongoing code development errors and provides higher accurate solutions automatically (or manually per request). But when comparing with others works, DevAssist is a combination of all of them with new era. DevAssist uses keywords, contextual key words and defined of the editor to provide higher accurate results in an intelligent manner.

## 7.3 Summary of the entire report

The entire report is stating about how a problem had raised for a solution, the background of the problem, find a solution for the problem, what is the solution, the background of the solution, addressing the solution, how that solution becomes a project, design the project with adding many enhancements, implement progress of the project and about the evaluation of the project.

#### **7.4 Summary**

In this chapter, it has discussed about the further works and how the system is differing from the others works. Further a summary about the entire project is included.

## Conclusion & Further work

### 8.1 Introduction

This Chapter will finalize the entire project and the report. In this chapter, there will be some references to the implementation of this plugin and discuss the matters with the outsider's perspective. In other words, here it will discuss the application in the commercial perspective rather than the technical perspective. This kind of a conclusion is needed to identify the barriers of the system that prevent it to be directly put in to the market.

At first, this chapter will discuss the achievement of the objectives, whether it is up to the expected performance. Then the problems that were encountered will be discussed. As the next step, the limitations of the provided solution will be discussed with the help of the evaluation carried out in the previous chapter. Finally, the further work will be discussed in order to complete the entire work proceeded up to now. Providing accurate results according to the percentage values of accuracy is important and catches commercial interests for context significant improvements of the software development industry. Therefore, DevAssist can be one of the most interesting plug-in among the developers hence its accuracy resulting giving nature and the sensible result collection to behave as an intelligent plug-in. It has given the options to input user's data manually as well as automatically.

### 8.2 Objectives

- Critically analyze and understand the problem
- Identify functional and non-functional requirements
- Design an overall architectural design
- Identify required technologies to address the issue
- Study the technologies
- Design and implement the system
- Develop a reliable solution

- Make recommendations to improve it further

### **8.3 Achievements of objectives**

Reliability is an important non-functional requirement which is an essential feature to develop a workable system. When developing system, it had concerned about getting the correct output for appropriate inputs. Wasting resource especially time is a common fact in many systems. But in this system, the waste has minimized a lot because it has integrated very important two tasks from one plug-in therefore the user does not want to take the use of any other application.

A system had developed by using latest Microsoft technologies and according to the currently available versions of Visual Studio. By user friendly interface of the system has been enriched the attractiveness. Further it has taken data from currently developing code editor therefore it increases the familiarity with user and the software. To check whether the system is efficient it had to do some tests on each unit and after the integration.

There were some more results emerged with the completion of the project. The vast area of knowledge that could be gained by doing this project was the one of the huge benefit that it could come across. By the beginning till the end of the project it had to learn most of the thing newly and apply them to the project. Therefore, from this final year project it could be gathered a huge knowledge on each part of the software development life cycle.

### **8.4 Problems Encountered**

It had to face for some problem because of the scarce of resources. The main problem was the lack of resources regarding to the natural language processing (NLP).

Further it happened to spend much time on researching how to generate accurate query for selected code snippet and how sites are rank because this is a new research area. One of my approaches is to identifying language related coding patterns e.g.: how variables are declared and identifying where it's used. We have to match every single pattern to separate those values from the selected code snippet. Developers are free to use any kind of standard to program. So, some are well formatted and some are not. I saw some developer declared variable without any meaning. E.g.: x, y, f\_name, l\_name . Second is how the retreated

URLs are ranked. Ranked URLs are match to selected code segment. There's no way to check the accuracy of the URLs. Without looking it we cannot guarantee the HTML content of web page. So we had to find good unique way to analyst and prioritize what was the best.

When doing a vast search on the research area there is lack of research papers return about these areas. Therefore, when doing the project, some areas had been done with minimum references.

### **8.5 Further work**

There are some further improvements for system as further works. Further works can be proposed for some input modules when considering the current functionalities.

Basically, the plug-in should be extended to other IDEs (Integrated Development Environment) as well. In other words, next step is IDE independent DevAssist plug-in development. The tool works both as a search query and selected code snippet, and helps the developers in solving their programming problems with providing code samples from top rated web sites. Especially it is associated with programming errors and exceptions. In future, I plan to conduct a more exhausted user study with prospective participants. Furthermore, increase the accuracy of query generating module and ranking algorithm. I also planned to extend this tool to support multiple languages and work with any other IDEs.

### **8.6 Summary**

Main objective of the project is to implement a best assistant tool for a developer. As of now there is no such good successful implementation. Available systems have more issues and drawbacks compared to DevAssist. Next chapter shows the references.



## References

- [1] K. Muşlu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, “Speculative analysis of integrated development environment recommendations,” *ACM SIGPLAN Not.*, vol. 47, no. 10, pp. 669–682, 2012.
- [2] C. Xia, J. Fielder, and L. Siragusa, “Achieving better peer interaction in online discussion forums: A reflective practitioner case study,” *Issues Educ. Res.*, vol. 23, no. 1, pp. 97–113, 2013.
- [3] A. X. Zhang, M. S. Ackerman, and D. R. Karger, “Mailing Lists: Why Are They Still Here, What’s Wrong With Them, and How Can We Fix Them?,” 2015, pp. 4009–4018.
- [4] A. Bacchelli, L. Ponzanelli, and M. Lanza, “Harnessing stack overflow for the ide,” in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, 2012, pp. 26–30.
- [5] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, “An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks,” *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, 2006.
- [6] I. Kwan, S. D. Fleming, and D. Piorkowski, “Information Foraging Theory for Collaborative Software Development,” 2012.
- [7] J. M. Fernández-Luna, J. F. Huete, R. Pérez-Vázquez, J. C. Rodríguez-Cano, and C. Shah, “COSME: A NetBeans IDE plugin as a team-centric alternative for search driven software development,” *CIS’10*, 2010.
- [8] M. M. Rahman and C. K. Roy, “Surfclipse: Context-Aware Meta-search in the IDE,” 2014, pp. 617–620.
- [9] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, “Mining StackOverflow to turn the IDE into a self-confident programming prompter,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 102–111.
- [10] Microsoft Corporation, “Overview of the .NET Framework.” .
- [11] Michael Gilleland, “Levenshtein Distance, in Three Flavors.” .

- [12] H. Saif, M. Fernandez, Y. He, and H. Alani, “On stopwords, filtering and data sparsity for sentiment analysis of Twitter,” 2014.
- [13] R. T.-W. Lo, B. He, and I. Ounis, “Automatically building a stopword list for an information retrieval system,” in *Journal on Digital Information Management: Special Issue on the 5th Dutch-Belgian Information Retrieval Workshop (DIR)*, 2005, vol. 5, pp. 17–24.
- [14] E. Loper and S. Bird, “NLTK: The natural language toolkit,” in *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, 2002, pp. 63–70.
- [15] Stack Exchange, Inc, “Stack Exchange API v2.2.” .
- [16] C. Treude and M. P. Robillard, “Augmenting API documentation with insights from stack overflow,” 2016, pp. 392–403.
- [17] Microsoft Corporation, “C# Keywords.”

## Appendix

### C# keywords and contextual keywords

C# Keywords

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	FALSE
finally	fixed	float	for
foreach	goto	if	implicit
in	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	TRUE	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

C# Contextual Keywords

add	alias	ascending
async	await	descending
dynamic	from	get
global	group	into
join	let	orderby
partial (type)	partial (method)	remove
select	set	value
var	where (generic type constraint)	where (query clause)
yield		

## Implementation of Query Generator

### Get code snippet from code editor

```
var m_dte = Microsoft.VisualStudio.Shell.Package.GetGlobalService(typeof(EnvDTE.DTE)) as EnvDTE.DTE;
    var _activeWindow = m_dte.ActiveWindow;
    var _ActiveDocument = m_dte.ActiveDocument;
    if (_ActiveDocument != null)
    {
        TextSelection _TextSelection = _ActiveDocument.Selection as TextSelection;

        if (_TextSelection != null)//user select 'devAssist' from main menu
            if (!string.IsNullOrEmpty(_TextSelection.Text))
                this.codeSnippet = _TextSelection.Text;
            else
                this.codeSnippet = DevAssistVSIX.V2.ToolWindow1Command.Instance
.selectedCodeSnippet;
    }
    DevAssistVSIX.V2.ToolWindow1Command.Instance.selectedCodeSnippet = string.Empty;

    if (!string.IsNullOrEmpty(codeSnippet))
    {
        ProcessQuery();
    }
}
```

### Making queries

```
private void QueryGen()
{
    if (string.IsNullOrEmpty(codeSnippet))
    {
        var m_dte = Microsoft.VisualStudio.Shell.Package.GetGlobalService(typeof(EnvDTE.DTE)) as EnvDTE.DTE;
        var _activeWindow = m_dte.ActiveWindow;
        var _ActiveDocument = m_dte.ActiveDocument;
        if (_ActiveDocument != null)
        {
            TextSelection _TextSelection = _ActiveDocument.Selection as TextSelection;
        }
    }
}
```

```

        if (_TextSelection != null)//user select 'devAssist' from main menu
            if (!string.IsNullOrEmpty(_TextSelection.Text))
                codeSnippet = _TextSelection.Text.Replace("@", string.Empty
    );
    }
}

if (!string.IsNullOrEmpty(codeSnippet))
{
    try
    {
        //remove comments
        var blockComments = @"\/\*(.*?)\*/";
        var lineComments = @"\/(.*?)\r?\n";
        var strings = @"\"\"((\\[^\n]|^[^\n]*)*)\"\"";
        var verbatimStrings = @"@\"\"([^\"]*)\"\"";
        var consoleWriteLine_ = @"Console.(.*?)\((.*?)\"";
        var betweenQuotes = new Regex(@"\".??\"");

        var valuebetweenQuotes = betweenQuotes.Matches(codeSnippet);

        foreach (var item in valuebetweenQuotes)//remove value between double
        quotes
        {
            codeSnippet = codeSnippet.Replace(item.ToString(), "");
        }

        string codeSnippetWithNoComments = Regex.Replace(codeSnippet, consoleWriteLine_ + "|" + blockComments + "|" + lineComments + "|" + strings + "|" + verbatimStrings, me =>
        {
            if (me.Value.StartsWith("/") || me.Value.StartsWith("//") || me.Value.StartsWith("Console."))
                return me.Value.StartsWith("//") ? Environment.NewLine : ""
        ;

            // Keep the literal strings
            return me.Value;
        }, RegexOptions.Singleline);

        //variable declarations
        var variabledeclare = @"[A-Za-z0-9]+(\s[A-Za-z0-9_]+)?";

```

```

        var variabledeclarewithValue = @"[A-Za-z0-9]+(\s[A-Za-z0-9_]+)+(\s[=]+)(\s[A-Za-z0-9._*]+)?";
        string variableNames = string.Empty;
        foreach (Match match in Regex.Matches(codeSnippetWithNoComments, variabledeclare))
        {
            if (match.Value.Split(' ').Length > 1)
            {
                variableNames += match.Value.Split(' ')[1].Replace(";", string.Empty) + ",";
            }
        }
        foreach (Match match in Regex.Matches(codeSnippetWithNoComments, variabledeclarewithValue))
        {
            if (match.Value.Split(' ').Length > 1)
            {
                variableNames += match.Value.Split(' ')[1].Replace(";", string.Empty) + ",";
            }
        }

        TermFrequency termFrequency = new TermFrequency();
        List<string> tokens = termFrequency.Tokenize(codeSnippetWithNoComments.ToLower());

        //spacial characters
        string specialChars = string.Empty;
        foreach (var item in tokens)
        {
            if (!Regex.IsMatch(item, @"^[a-zA-Z0-9_]+$"))
                specialChars += item + ",";
        }
        string onlynumbers = string.Empty;
        foreach (var item in tokens)
        {
            if (Regex.IsMatch(item, @"^-*[0-9,\.\.]+$"))
                onlynumbers += item + ",";
        }
        //remove variables + onlynumbers + specialChars

```

```

        foreach (var item in onlynumbers.Split(',').Concat(variableNames.Split(',')).Concat(specialChars.Split(',')))
        {
            tokens.RemoveAll(x => x == item.ToLower());
        }

        StreamReader _keywords;
        StreamReader _contextualkeywords;

        List<string> keywordMatch = new List<string>();
        List<string> contextualKeywordMatch = new List<string>();
        List<string> keywordNotMatch = new List<string>();
        List<string> contextualKeywordNotMatch = new List<string>();//put t
o 1 array

        var keywords = keywordsData.Split(',').Select(p => p.Trim()).ToArray(); // _keywords.ReadToEnd().Replace("\r\n", string.Empty).Split(',');
        foreach (var word in tokens)
        {
            if (keywords.Contains(word.Trim()))
            {
                keywordMatch.Add(word);
            }
            else
                keywordNotMatch.Add(word.Trim());
        }

        var contextualkeywords = contextualKeywordData.Split(',').Select(p
=> p.Trim()).ToArray();
        foreach (var word in keywordNotMatch)
        {
            if (contextualkeywords.Contains(word.Trim()))
            {
                contextualKeywordMatch.Add(word.Trim());
            }
            else
                contextualKeywordNotMatch.Add(word.Trim());
        }
        // }
        //TERM FREQUENCY METHOD-----
-----

```

```

List<string> remainingTokens = contextualKeywordNotMatch;
List<string> _contextualKeywordList = contextualKeywordMatch;
List<string> _keywordList = keywordMatch;
List<string> _variables = variableNames.Split(',').ToList();
//contextual keyword TF
//keyword TF
//variable names TF
Dictionary<string, int> tfTokens;
Dictionary<string, int> sortedTFTokens = null;

if (remainingTokens.Count > 2)
{
    tfTokens = termFrequency.ToStrIntDict(remainingTokens.ToArray());
    sortedTFTokens = termFrequency.ListWordsByFreq(tfTokens, TermFr
equency.SortOrder.Descending);
}
else
{
    if (_contextualKeywordList.Count > 2)
    {
        tfTokens = termFrequency.ToStrIntDict(_contextualKeywordLis
t.ToArray());
        if (remainingTokens.Count > 0 && !string.IsNullOrEmpty(remain
gTokens[0]))
            tfTokens.Add(remainingTokens[0], 100); //add remaining token
and prioritize as 1st
        if (remainingTokens.Count > 1 && !string.IsNullOrEmpty(remain
gTokens[1]))
            tfTokens.Add(remainingTokens[1], 99); //add remaining token
and prioritize as 2nd
        sortedTFTokens = termFrequency.ListWordsByFreq(tfTokens, Te
rmFrequency.SortOrder.Descending);
    }
    else
    {
        if (_keywordList.Count > 2)
        {
            tfTokens = termFrequency.ToStrIntDict(_keywordList.ToAr
ray());

```



```

        if (remaingTokens.Count > 0 && !string.IsNullOrEmpty(re
maingTokens[0]))
            tfTokens.Add(remaingTokens[0], 100); //add remaing t
oken and prioritize as 1st
        if (remaingTokens.Count > 1 && !string.IsNullOrEmpty(re
maingTokens[1]))
            tfTokens.Add(remaingTokens[1], 99); //add remaing to
ken and prioritize as 2nd
        if (_contextualKeywordList.Count > 0 && !string.IsNullo
rEmpty(_contextualKeywordList[0]))
            tfTokens.Add(_contextualKeywordList[0], 98); //add c
ontextial keyword and prioritize as 3rd
        if (_contextualKeywordList.Count > 1 && !string.IsNullo
rEmpty(_contextualKeywordList[1]))
            tfTokens.Add(_contextualKeywordList[1], 97); //add c
ontextial keyword and prioritize as 4th
        sortedTFTokens = termFrequency.ListWordsByFreq(tfTokens
, TermFrequency.SortOrder.Descending);
    }
}
}
//get first 5 items from dictionary to build the query
string buildedQuery = string.Empty;
if (sortedTFTokens != null)
{
    if (sortedTFTokens.Count > 4)
    {
        int count = 0;
        foreach (var item in sortedTFTokens)
        {
            if (count == 5)
                break;
            buildedQuery += item.Key + " ";
            count++;
        }
    }
    else
        foreach (var item in sortedTFTokens)
            buildedQuery += item.Key + " ";
}
if (buildedQuery != string.Empty || buildedQuery != " ")

```

```

        {
            searchKey = buildedQuery.Replace("\"", string.Empty).Replace("\
", string.Empty);
            this.txtSearch.Text = buildedQuery.Replace("\"", string.Empty).
Replace("\", string.Empty);
            Button_Click(new object(), new RoutedEventArgs());
        }
        else
        {
            tbSearching.Text = "Query Build failed";
        }
    }
    catch (Exception ex)
    {
        tbSearching.Text = ex.Message;//log
    }
    finally
    {
        codeSnippet = string.Empty;
        DevAssistVSIX.V2.ToolWindow1Command.Instance.selectedCodeSnippet =
string.Empty;
        tbSearching.Text = string.Empty;
    }
}
else
    tbSearching.Text = "highlight code segment and click to auto generate "
;
}

```

## Tokenize and Sort

```

public enum SortOrder
{
    Ascending, // from small to big numbers or alphabetically.
    Descending // from big to small number or reversed alphabetical order
}
// This will discard digits
private static char[] delimiters_no_digits = new char[] {
    '{', '}', '(', ')', '[', ']', '>', '<', '-', '_', '=', '+',
    '|', '\\', ':', ';', ' ', ',', '.', '/', '?', '~', '!',

```

```

        '@', '#', '$', '%', '^', '&', '*', ' ', '\n', '\n', '\t',
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

public List<string> Tokenize(string text)
{
    string[] tokens = text.Split(delimiters_no_digits, StringSplitOptions.RemoveEmptyEntries);
    tokens = tokens.ToList().Where(p => p.Length > 1).ToArray();//length more than 1

    for (int i = 0; i < tokens.Length; i++)
    {
        string token = tokens[i];

        // Change token only when it starts and/or ends with "" and
        // it has at least 2 characters.
        token = token.Replace(System.Environment.NewLine, string.Empty).Replace("@", string.Empty);
        if (token.Length > 1)
        {
            if (token.StartsWith("")) && token.EndsWith(""))
                tokens[i] = token.Substring(1, token.Length - 2); // remove the starting and ending ""

            else if (token.StartsWith(""))
                tokens[i] = token.Substring(1); // remove the starting ""

            else if (token.EndsWith(""))
                tokens[i] = token.Substring(0, token.Length - 1); // remove the last ""
        }
    }

    return tokens.ToList();
}

public Dictionary<string, int> ToStrIntDict(string[] words)
{
    Dictionary<string, int> dict = new Dictionary<string, int>();

    foreach (string word in words)

```

```

    {
        // if the word is in the dictionary, increment its freq.
        if (dict.ContainsKey(word))
        {
            dict[word]++;
        }
        // if not, add it to the dictionary and set its freq = 1
        else
        {
            dict.Add(word, 1);
        }
    }

    return dict;
}

public Dictionary<string, int> ListWordsByFreq(Dictionary<string, int> strIntDict,
SortOrder sortOrder)
{
    // Copy keys and values to two arrays
    string[] words = new string[strIntDict.Keys.Count];
    strIntDict.Keys.CopyTo(words, 0);

    int[] freqs = new int[strIntDict.Values.Count];
    strIntDict.Values.CopyTo(freqs, 0);

    //Sort by freqs: it sorts the freqs array, but it also rearranges
    //the words array's elements accordingly (not sorting)
    Array.Sort(freqs, words);

    // If sort order is descending, reverse the sorted arrays.
    if (sortOrder == SortOrder.Descending)
    {
        //reverse both arrays
        Array.Reverse(freqs);
        Array.Reverse(words);
    }

    //Copy freqs and words to a new Dictionary<string, int>
    Dictionary<string, int> dictByFreq = new Dictionary<string, int>();

```

```

    for (int i = 0; i < freqs.Length; i++)
    {
        dictByFreq.Add(words[i], freqs[i]);
    }

    return dictByFreq;
}

```

## Implementation of Search Service

```

private void Search()
{
    string query = searchKey;
    WebSearchManager _search = new WebSearchManager();
    APISearchManager _ApiSearch = new APISearchManager();

    List<TechUrl> yahoo = _search.SearchQuery("google", query).Where(p => p.Browser == "google").GroupBy(p => p.Url).Select(p => p.First()).ToList();//encord url
    List<TechUrl> yahoo = _search.SearchQuery("yahoo", query).Where(p => p.Browser == "yahoo").GroupBy(p => p.Url).Select(p => p.First()).ToList();//encord url
    List<TechUrl> bing = _search.SearchQuery("bing", query).Where(p => p.Browser == "bing").GroupBy(p => p.Url).Select(p => p.First()).ToList();//encord url

    margeResults = yahoo.Concat(bing).GroupBy(p => p.Url).Select(p => p.First()).ToList();//l //.Where(p => p.Description.Contains(languageDefault)

    int id = 0;
    removeFiles();//previously saved files
    foreach (var listItem in margeResults)
    {
        listItem.id = id++;
        listItem.PageTitle = WebSearchManager.removeTags(WebSearchManager.wordReplace(_search.LoadSinglePage(listItem.id, listItem.Url)));
    }

    margeResults.RemoveAll(p => p.PageTitle.Length < 3);//remove short title links

    List<Score> _scores = new List<Score>();
}

```

```

for (int i = 0; i < margeResults.Count - 1; i++)
{
    for (int j = i + 1; j < margeResults.Count; j++)
    {
        double WordsRatio;
        double RealWordsRatio;
        double score;
        score = _search.SimilarityTwoString(margeResults[i].PageTitle, margeResults[j].PageTitle, out WordsRatio, out RealWordsRatio);
        _scores.Add(new Score() { ID = margeResults[i].id, Value = score })
;
    }
}

var _ids = _scores.GroupBy(c => c.ID)
    .Select(grp => new
    {
        grp.Key,
        MaxValue = grp.OrderByDescending(
            x => x.Value).FirstOrDefault()
    }).ToList();

for (int i = 0; i < _ids.Count; i++)
{
    if (margeResults[i].id == _ids[i].MaxValue.ID)
        margeResults[i].Score = _ids[i].MaxValue.Value;
}

margeResults = margeResults.OrderByDescending(p => p.Score).ToList();
}

```

## Filter Urls

```

public List<TechUrl> SearchQuery(string browser, string query)
{
    List<TechUrl> __TechUrls = new List<TechUrl>();
    webclient = new WebClient();

    if (browser == "yahoo")

```

```

    {
        string htmlContent = webclient.DownloadString(yahoo + System.Uri.Escape
DataString(query));
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    else if (browser == "bing")
    {
        string htmlContent = webclient.DownloadString(bing + System.Uri.Escaped
ataString(query));
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    else if (browser == "google")
    {
        string htmlContent = webclient.DownloadString(google + query);
        __TechUrls = FilterUrl(browser, htmlContent);
    }
    return __TechUrls;
}

```

## Filter Title Meta data and Code snippets

```

string patternTitle = "(?i)<title>(.*?)</title>";
string patternMetaTags = "(?i)<meta([^\>]+)>";
string patternCodeSnippetTags = "(?i)<pre>(.*?)</pre>"; //<CODE> / <PRE> / <SAMP>
string patternCodeSnippetTags2 = "(?i)<pre([^\>]+)>(.*?)</pre>"; //<CODE> / <PRE> / <SAMP
>

```

```

private void FilterCodeSnippets(int id, string htmlContent)
{
    //stackoverflow "acceptedAnswer" "answer accepted-answer"
    Console.WriteLine("-----");

    //---conditions -- stackoverflow-----
    string stackoverflowAnswerDiv = "(?i)<div([^\>]+)class=\"answer([^\>+)]\"([^\>
]+)>(.\|\\n)*?</body>";
    MatchCollection _AnswerDivs = Regex.Matches(htmlContent, stackoverflowAnsw
erDiv, RegexOptions.Singleline);
    bool voteAcceptOn = htmlContent.Contains("vote-accepted-
on"); //only for stackoverflow

```

```

        MatchCollection _titleMetaContents = Regex.Matches(_AnswerDivs.Count > 0 ?
        _AnswerDivs[0].Value : htmlContent, patternCodeSnippetTags, RegexOptions.Singleline);
        if (_titleMetaContents.Count < 1)
            _titleMetaContents = Regex.Matches(htmlContent, patternCodeSnippetTags2
, RegexOptions.Singleline);
        StringBuilder _content = new StringBuilder();
        if (_titleMetaContents.Count > 0)
        {
            _content.Append(voteAcceptOn ? "<hr style=\"height:4px; border: none; c
olor: green; background-color:green; \">" : "<hr/>");

            foreach (var snippet in _titleMetaContents)
            {
                _content.Append(snippet);
                _content.Append("<hr/>");
            }
            using (StreamWriter _file = new StreamWriter(@path + "DevAssistsTemp\\w
eb_content_dev_ast_pre_tag_" + id + ".txt"))
            {
                _file.Write(_content.ToString());
            }
        }
    }
    private string FilterTitleAndMeta(string htmlContent, bool title = true)
    {
        Console.WriteLine("-----");
        MatchCollection _titleMetaContents = Regex.Matches(htmlContent, title ? pat
ternTitle : patternMetaTags, RegexOptions.Singleline);
        if (_titleMetaContents.Count > 0)
            return removeTags(wordReplace(_titleMetaContents[0].Value));
        return string.Empty;
    }
}

```

## Implementation of Ranking Model

```

public void Ranking(){
    for (int i = 0; i < margeResults.Count - 1; i++)
    {

```



```

        for (int j = i + 1; j < margeResults.Count; j++)
        {
            double WordsRatio;
            double RealWordsRatio;
            double score;
            score = _search.SimilarityTwoString(margeResults[i].PageTitle, margeResults[j].PageTitle, out WordsRatio, out RealWordsRatio);
            _scores.Add(new Score() { ID = margeResults[i].id, Value = score });
        }
    }

    var _ids = _scores.GroupBy(c => c.ID)
        .Select(grp => new
        {
            grp.Key,
            MaxValue = grp.OrderByDescending(
                x => x.Value).FirstOrDefault()
        }).ToList();

    for (int i = 0; i < _ids.Count; i++)
    {
        if (margeResults[i].id == _ids[i].MaxValue.ID)
            margeResults[i].Score = _ids[i].MaxValue.Value;
    }

    margeResults = margeResults.OrderByDescending(p => p.Score).ToList(); //check orderby
}

```

### Get Similarity Ratio of each Title and Meta description

```

public double SimilarityTwoString(String FullString1, String FullString2, out double WordsRatio, out double RealWordsRatio)
{
    double theResult = 0;
    String[] Splitted1 = FullString1.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    String[] Splitted2 = FullString2.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
    if (Splitted1.Length < Splitted2.Length)
    {

```

```

        String[] Temp = Splitted2;
        Splitted2 = Splitted1;
        Splitted1 = Temp;
    }
    int[,] theScores = new int[Splitted1.Length, Splitted2.Length]; //Keep the best scores for each word. 0 is the best, 1000 is the starting.
    int[] BestWord = new int[Splitted1.Length]; //Index to the best word of Splitted2 for the Splitted1.

    for (int loop = 0; loop < Splitted1.Length; loop++)
    {
        for (int loop1 = 0; loop1 < Splitted2.Length; loop1++) theScores[loop, loop1] = 1000;
        BestWord[loop] = -1;
    }
    int WordsMatched = 0;
    for (int loop = 0; loop < Splitted1.Length; loop++)
    {
        String String1 = Splitted1[loop];
        for (int loop1 = 0; loop1 < Splitted2.Length; loop1++)
        {
            String String2 = Splitted2[loop1];
            int LevenshteinDistance = Compute(String1, String2);
            theScores[loop, loop1] = LevenshteinDistance;
            if (BestWord[loop] == -1 || theScores[loop, BestWord[loop]] > LevenshteinDistance) BestWord[loop] = loop1;
        }
    }

    for (int loop = 0; loop < Splitted1.Length; loop++)
    {
        if (theScores[loop, BestWord[loop]] == 1000) continue;
        for (int loop1 = loop + 1; loop1 < Splitted1.Length; loop1++)
        {
            if (theScores[loop1, BestWord[loop1]] == 1000) continue; //the worst score available, so there are no more words left
            if (BestWord[loop] == BestWord[loop1]) //2 words have the same best word
            {
                //The first in order has the advantage of keeping the word in e
                quality
            }
        }
    }

```

```

    if (theScores[loop, BestWord[loop]] <= theScores[loop1, BestWord[loop1]])
    {
        theScores[loop1, BestWord[loop1]] = 1000;
        int CurrentBest = -1;
        int CurrentScore = 1000;
        for (int loop2 = 0; loop2 < Splitted2.Length; loop2++)
        {
            //Find next bestword
            if (CurrentBest == -1 || CurrentScore > theScores[loop1, loop2])
            {
                CurrentBest = loop2;
                CurrentScore = theScores[loop1, loop2];
            }
            BestWord[loop1] = CurrentBest;
        }
        else//the latter has a better score
        {
            theScores[loop, BestWord[loop]] = 1000;
            int CurrentBest = -1;
            int CurrentScore = 1000;
            for (int loop2 = 0; loop2 < Splitted2.Length; loop2++)
            {
                //Find next bestword
                if (CurrentBest == -1 || CurrentScore > theScores[loop, loop2])
                {
                    CurrentBest = loop2;
                    CurrentScore = theScores[loop, loop2];
                }
                BestWord[loop] = CurrentBest;
            }
        }
        loop = -1;
        break;//recalculate all
    }
}
}

```

```

        for (int loop = 0; loop < Splitted1.Length; loop++)
        {
            if (theScores[loop, BestWord[loop]] == 1000) theResult += Splitted1[loop
p].Length; //All words without a score for best word are max failures
            else
            {
                theResult += theScores[loop, BestWord[loop]];
                if (theScores[loop, BestWord[loop]] == 0) WordsMatched++;
            }
        }
        int theLength = (FullString1.Replace(" ", "").Length > FullString2.Replace(
" ", "").Length) ? FullString1.Replace(" ", "").Length : FullString2.Replace(" ", "").L
ength;

        if (theResult > theLength) theResult = theLength;
        theResult = (1 - (theResult / theLength)) * 100;
        WordsRatio = ((double)WordsMatched / (double)Splitted2.Length) * 100;
        RealWordsRatio = ((double)WordsMatched / (double)Splitted1.Length) * 100;
        return theResult;
    }

private int Compute(string s, string t) //LevenshteinDistance
{
    int n = s.Length;
    int m = t.Length;
    int[,] d = new int[n + 1, m + 1];

    // Step 1
    if (n == 0)
    {
        return m;
    }

    if (m == 0)
    {
        return n;
    }

    // Step 2
    for (int i = 0; i <= n; d[i, 0] = i++)
    {
    }
}

```

```

for (int j = 0; j <= m; d[0, j] = j++)
{
}

// Step 3
for (int i = 1; i <= n; i++)
{
    //Step 4
    for (int j = 1; j <= m; j++)
    {
        // Step 5
        int cost = (t[j - 1] == s[i - 1]) ? 0 : 1;

        // Step 6
        d[i, j] = Math.Min(
            Math.Min(d[i - 1, j] + 1, d[i, j - 1] + 1),
            d[i - 1, j - 1] + cost);
    }
}
// Step 7
return d[n, m];
}

```