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)) a
s EnvDTE.DTE;
            var _activeWindow = m_dte.ActiveWindow;
            var _ActiveDocument = m_dte.ActiveDocument;
            if (_ActiveDocument != null)
            {
                TextSelection _TextSelection = _ActiveDocument.Selection as TextSelecti
on;
                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.E
mpty;

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

Making queries

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

```
                    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 doub
le quotes
                    {
                        codeSnippet = codeSnippet.Replace(item.ToString(), "*");
                    }

                    string codeSnippetWithNoComments = Regex.Replace(codeSnippet, conso
leWriteLine_ + "|" + blockComments + "|" + lineComments + "|" + strings + "|" + verbati
mStrings, me =>
                    {
                        if (me.Value.StartsWith("/*") || me.Value.StartsWith("//") || m
e.Value.StartsWith("Console."))
                            return me.Value.StartsWith("//") ? Environment.NewLine : ""
;
                        // Keep the literal strings
                        return me.Value;
                    }, RegexOptions.Singleline);

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

```
                    var varibledeclarewithValue = @"[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, va
ribledeclare))
                    {
                        if (match.Value.Split(' ').Length > 1)
                        {
                            variableNames += match.Value.Split(' ')[1].Replace(";", str
ing.Empty) + ",";
                        }
                    }
                    foreach (Match match in Regex.Matches(codeSnippetWithNoComments, va
ribledeclarewithValue))
                    {
                        if (match.Value.Split(' ').Length > 1)
                        {
                            variableNames += match.Value.Split(' ')[1].Replace(";", str
ing.Empty) + ",";
                        }
                    }

                    TermFrequency termFrequency = new TermFrequency();
                    List<string> tokens = termFrequency.Tokenize(codeSnippetWithNoComme
nts.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 + onlyumbers + specialChars
```

```
                    foreach (var item in onlynumbers.Split(',').Concat(variableNames.Sp
lit(',')).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()).ToArra
y(); //_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> 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, TermFr
equency.SortOrder.Descending);
                    }
                    else
                    {
                        if (_contextualKeywordList.Count > 2)
                        {
                            tfTokens = termFrequency.ToStrIntDict(_contextualKeywordLis
t.ToArray());
                            if (remaingTokens.Count > 0 && !string.IsNullOrEmpty(remain
gTokens[0]))
                                tfTokens.Add(remaingTokens[0], 100);//add remaing token
 and prioratize as 1st
                            if (remaingTokens.Count > 1 && !string.IsNullOrEmpty(remain
gTokens[1]))
                                tfTokens.Add(remaingTokens[1], 99);//add remaing token
and prioratize 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 prioratize as 1st
                                if (remaingTokens.Count > 1 && !string.IsNullOrEmpty(re
maingTokens[1]))
                                    tfTokens.Add(remaingTokens[1], 99);//add remaing to
ken and prioratize as 2nd
                                if (_contextualKeywordList.Count > 0 && !string.IsNullO
rEmpty(_contextualKeywordList[0]))
                                    tfTokens.Add(_contextualKeywordList[0], 98);//add c
ontextial keyword and prioratize as 3rd
                                if (_contextualKeywordList.Count > 1 && !string.IsNullO
rEmpty(_contextualKeywordList[1]))
                                    tfTokens.Add(_contextualKeywordList[1], 97);//add c
ontextial keyword and prioratize 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[] {
            '{', '}', '(', ')', '[', ']', '>', '<','-', '_', '=', '+',
            '|', '\\', ':', ';', ' ', ',', '.', '/', '?', '~', '!',
```

```
            '@', '#', '$', '%', '^', '&', '*', ' ', '\r', '\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.Remov
eEmptyEntries);
            tokens = tokens.ToList().Where(p => p.Length > 1).ToArray();//length more t
hen 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> strIntDi
ct, 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.Bro
wser == "google").GroupBy(p => p.Url).Select(p => p.First()).ToList();//encord url
            List<TechUrl> yahoo = _search.SearchQuery("yahoo", query).Where(p => p.Brow
ser == "yahoo").GroupBy(p => p.Url).Select(p => p.First()).ToList();//encord url
            List<TechUrl> bing = _search.SearchQuery("bing", query).Where(p => p.Browse
r == "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.wordR
eplace(_search.LoadSinglePage(listItem.id, listItem.Url)));
            }

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

            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, marg
eResults[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, stackoverflowAnswe
rDiv, 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 voide 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, marg
eResults[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(); //che
ck orderby
}
```

Get Similarity Ratio of each Title and Meta description

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

```
                String[] Temp = Splitted2;
                Splitted2 = Splitted1;
                Splitted1 = Temp;
            }
            int[,] theScores = new int[Splitted1.Length, Splitted2.Length];//Keep the b
est scores for each word.0 is the best, 1000 is the starting.
            int[] BestWord = new int[Splitted1.Length];//Index to the best word of Spli
tted2 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, BestWor
d[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[loo
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];
}
```