**Establishing Test Case and Change Request Relationship through Similarity Analysis**

JOHN EMERSON FERREIRA REGIS FILHO (STUDENT)

Informatic Center - UFPE, [johnemerson67@gmail.com](mailto:johnemerson67@gmail.com)

RICARDO BASTOS CAVALCANTE PRUDÊNCIO (ADVISOR)

Informatic Center - UFPE, [rbcp@cin.ufpe.br](mailto:rbcp@cin.ufpe.br)

JOSÉ CARLOS DO NASCIMENTO (PRODUCT OWNER)

Institute of Development and Technology - INDT, [josern@motorola.com](mailto:josern@motorola.com)

In the context of Motorola software testing, a lot of failures and change requests (CRs) are reported everyday. Sometimes an open CR does not correspond to real defects. In some cases, it can be just a feature which has changed its behavior but is working as designed (WAD). As the testers do not have any documentation, such cases are wrongly reported in CRs as possible defects. The analysis of those CRs is made manually and refactoring can only be made in the test case that was mentioned in the CR. In this situation, the analysis misses other test cases with wrong or obsolete steps that should be refactored as well. In this work, we propose a smart solution to automate the process of identifying missing test cases. This is done by measuring the similarity between WAD CRs and existing test cases, thus ranking potential related test cases for refactoring. By employing this approach, we aim to efficiently identify test cases that may have been missed and ensure that these test cases are updated.

**KEYWORDS:**  
Cosine Similarity, Change Request, Refactoring

CCS Concepts: • Software and its engineering, Software testing and refactor

**1 INTRODUCTION**

The society we live in is highly dependent on cell phones in our daily lives. Hence, it is crucial to produce them with high quality. An effective way to ensure the product quality is through testing, which is the main activity carried out by the CIn-Motorola project. Specifically, the CIn-Motorola project conducts testing activities during different parts of the life cycle of the software.

During the software life cycle, a lot of changes occur due to requirement updates, bug fixes and changes of behavior. The software changes, such as code change, implementation of new features, behavior change are expensive and spend more than 50% of the budget of the project [1]. Some changes in the code involve: 1) understand how they affect the software; 2) implement the proposed features; and 3) testing the changed system [2].  
 The development of a system includes lots of activities which despite all the techniques that are known these days, are still vulnerable to errors. To ensure the quality of products, some activities are introduced in the development process, including VV&T (verification, validation and testing) activities to minimize the occurrence of defects and associated risks. Among these activities, testing is one of the most frequently used. In addition to other activities such as the use of revisions and formal and rigorous specification and verification techniques, testing improves the reliability of the software.

In order to improve the efficiency and reliability of software development and testing, the investigation of new tools and techniques to optimize testing procedures is crucial. In the CIn-Motorola, once an issue is found by a tester, a Change Request (CR) report is opened, potentially identifying a software bug. However, due to knowledge gaps, some issues observed by a tester may be just a feature change, and not an actual bug. In this case, during a post-analysis process, the CR is assigned with the status working as design (WAD), which can result in a refactoring of the Test Case (TC) related to the CR.

The purpose of this research is to create a solution to retrieve TCs that are candidates to be refactored but missed in the CR description. This is done by measuring the similarity between the content of WAD CRs and the available TCs. In this analysis, we adopted natural language techniques to measure similarity between attribute vectors that represent: (1) the comments and description of CRs and (2) the steps and expected results of TCs. In this way, it will be possible to reduce the effort to open new CRs that will be closed as WAD and increase the TCs which are refactored, making it possible to gain time with the tests.

**2 BACKGROUND**

In this section, the life cycle of CRs will be presented (Section 2.1), followed by a definition of bad CRs (Section 2.2). Finally we present how the analysis of a WAD CR is done (Section 2.3).

**2.1 Bug Reports and Change Requests**

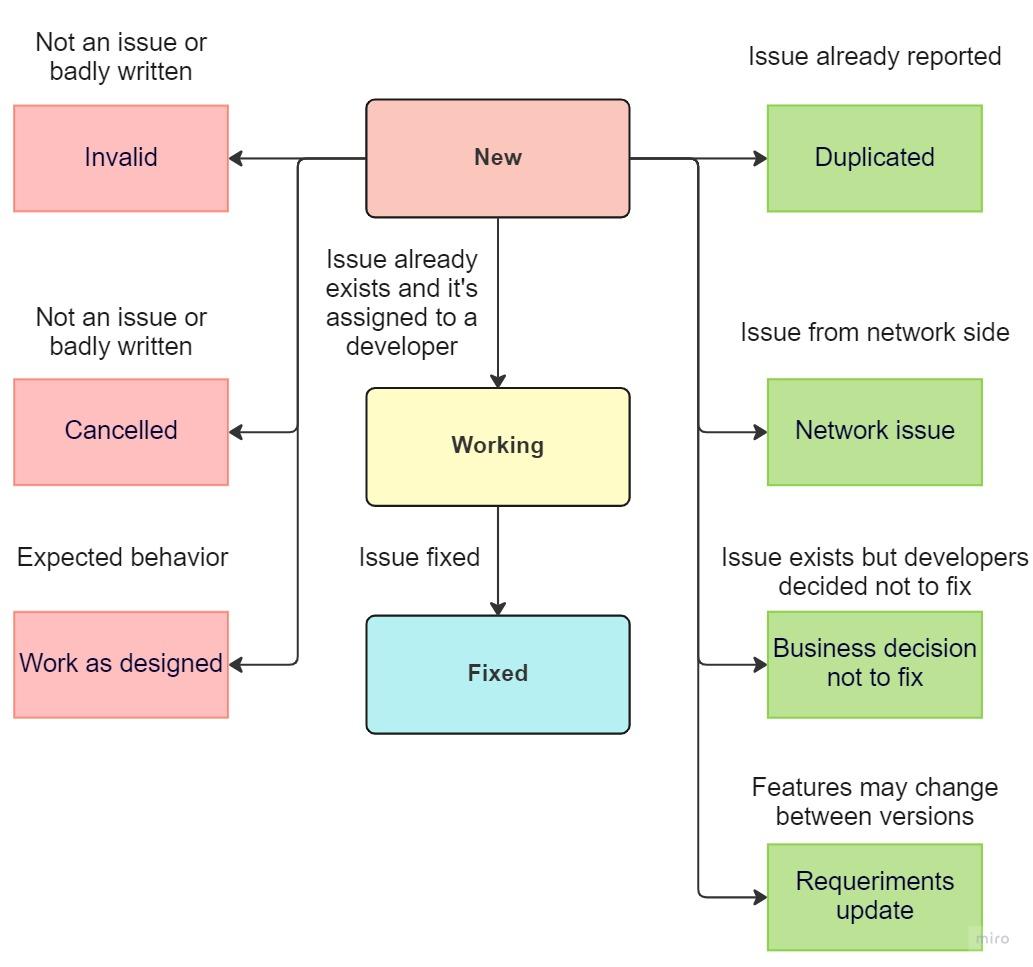


Fig. 1. Graph of the life cycle of a Change Request

A software bug refers to an error, flaw or fault in a computer program or system that leads to an unexpected behavior or outcome [3]. It is worth noting that there is a difference between a "bug" and an "issue". While an issue can be a bug, it can also take the form of a feature request, a task, documentation, among others.

Within the context of CIn-Motorola, the term "issue" is more commonly used than "bug" because during the Quality Assurance (QA) process, testers need to be aware of factors, other than bugs, that could potentially lead to a poor user experience. Consequently, in this scenario, testers open Change Requests (CRs) to notify developers of potential problems and request changes in the software.

A CR status changes during its existence, being assigned to different resolution states. This life-cycle can be seen in Figure 1. When a CR is opened, it is assigned to a status of NEW by the reviewer. The CR is then forwarded to a triage team, which reviews several factors related to the request. If a similar issue has already been reported, CR is closed with a DUPLICATED resolution status. If not, the request is forwarded to the team responsible for the feature in question and a developer is assigned to it. If the developer verifies that the issue is not valid, CR is closed with a INVALID, CANCELED or WORK AS DESIGNED resolution. If the developer identifies an issue and starts working on it, the status is changed to WORKING. Once the bug is fixed in the system, CR is closed with a status of FIXED. Occasionally, a CR is closed for reasons such as NETWORK ISSUE, REQUIREMENTS UPDATE or BUSINESS DECISION NOT TO FIX.

**2.2 Bad CRs**

CRs have a lifecycle, reaching in the end with a resolution considered as *bad CR*. There are three types of resolutions that can be considered bad: Invalid, Canceled and Work as designed. Invalid and Canceled ones are considered bad because they are either poorly written or not a problem. Work as designed is considered bad because the software behaves as expected, but the tester reports the observed behavior as an issue.

Each month, the Common Validation Team conducts an assessment of bad CRs to evaluate the team's effectiveness and to identify any feature changes. This analysis plays a crucial role in improving the effectiveness of opening CRs. However, the current process involves manually entering data into a spreadsheet as seen in Figure 2, which can be time-consuming for testers. Ideally, the tester who created the CR should perform this analysis, but not all testers are able to do so. Despite these challenges, the team is committed to continuously improving the review process.

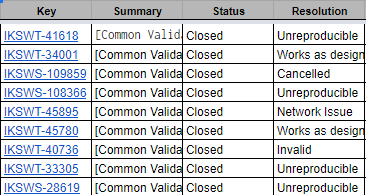


Fig. 2: Some bad CRs of Commom Validation Team

**2.3 Analysis of CRs WAD**

In the process to classify a CR as WAD, the tester must open the CR and should indicate which TC has failed on the field “Additional information -> Dalek test case” (see Figure 3). Then a developer will check the information and see if the software behavior is expected or not. There are many ways to do this, such as checking documentation, Pixel phone, looking for features changes, among others. After that, the developer will reference the document in the CR comments and close the CR as WAD. Finally, the tester will make a request to refactor the TC, more specifically, the part which was indicated in the CR as work as designed.

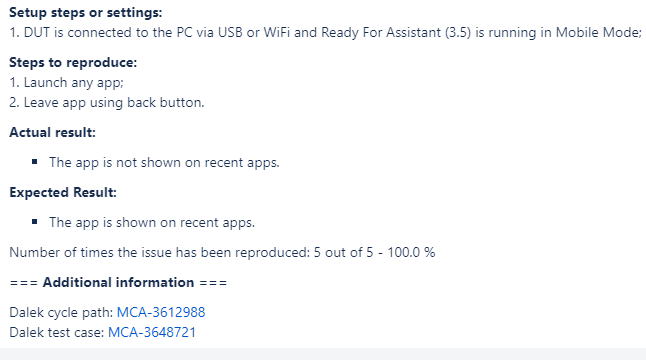


Fig. 3: Example of the content of a CR

Besides the TC indicated in the CR, other test cases may have coverage for a certain feature that had its behavior changed. Such TCs may have steps and results that are different from what they really should have, since the refactoring action is only required for the TC indicated in CR. If such related TCs with change in behavior are not refactored they can result in opening new CRs in the future that will be closed again with the WAD status. This is the drawback that motivates this work. The answer to this problem helps to increase the number of refactored test cases, consequently decreasing the number of WAD CRs in the future as well.

**3 METHODOLOGY AND MATERIALS**

**3.1 Objectives**

When a CR is open, the TC which failed is usually cited on the description of the CR. Then when its resolution is classified as WAD, it requires an action of refactoring the TC. This process has a limitation because not only the TC cited can be refactored. Other TCs related to the CR but not cited on it (e.g related to the same feature), may require a refactor.

To minimize the above limitation, we propose a process to retrieve TCs from the description of the WAD CRs. The objective is to increase the scope of TCs that need to be updated, and then decrease the number of WAD CRs potentially opened to those obsolete TCs.

**3.2 Development stages**

The development was divided into a few steps, such as data capture (CRs and TCs), database pre-processing, data modeling and analysis. The solution flow is described as follows (Figure 4). Given a WAD CR, the proposed solution performs a search for similarity between the content of each CR and the content of each TC. The goal is to find potential TCs that can generate new WAD CRs. For this, initially both the textual content of the CR and the TC is pre-processed (see section 3.3). Then, in the Data Modeling stage, a vector representation of the CR and the TC is created (see section 3.4).

Then the similarity is calculated between those representations. In case of the threshold being higher than 0.5 (see section 3.4), the TC should pass through a manual analysis for refactoring. The tester must decide if the TC should be refactored or not.

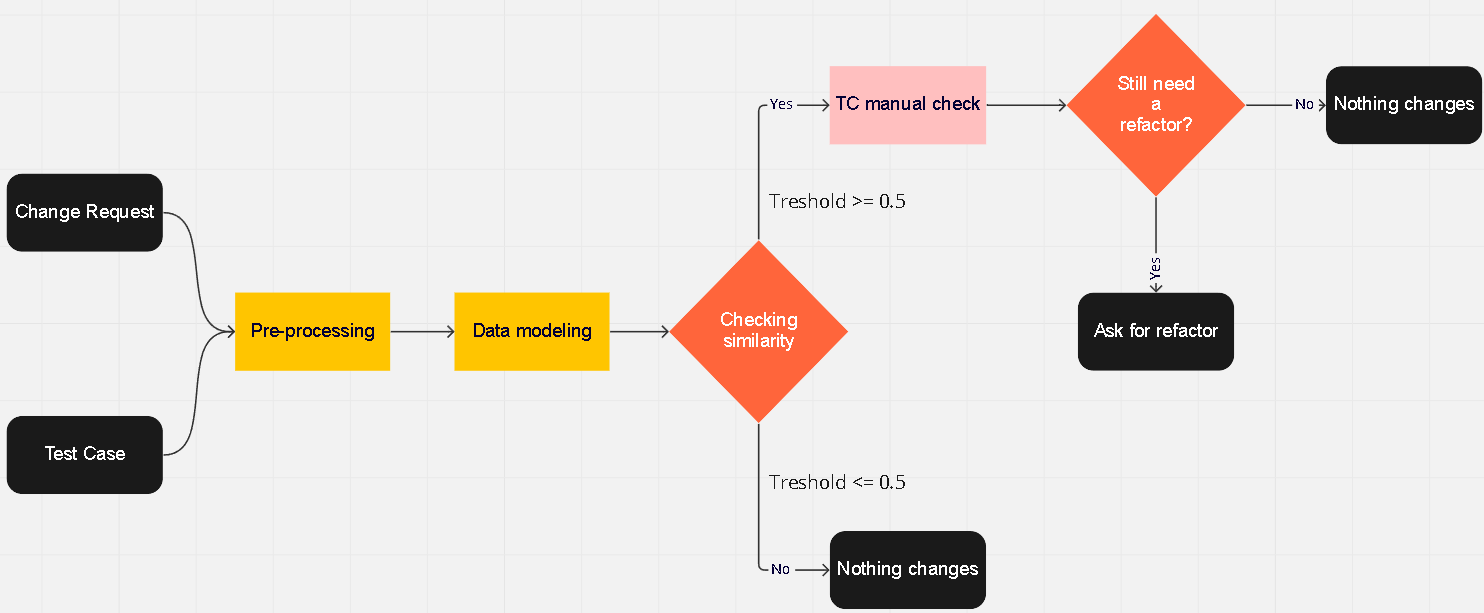


Fig. 4: Development stages

**3.3 Text preprocessing**

With the exception of IDs for both CRs and TCs, all other fields such as the description of CR and initial setup, steps, and expected results of the TCs have gone through the preprocessing step. The CRs and TCs had unnecessary information like punctuation, numbers and stopwords. With this, the pre-processing function removed these unnecessary characters, in order to make the text cleaner for future verification of similarity. Text preprocessing is a crucial step in natural language processing where text data is cleaned and prepared before analysis or modeling [4].

**3.3.1 Tokenization**

The tokenization process was employed to break down a sentence into a sequence of strings separated into keywords, while maintaining the original meaning of the text or extracted sentence [5]. In this particular project, the description of the CRs, along with the initial setup, steps, and expected results, of the TCs, were subjected to tokenization. Figure 5 illustrates a tokenization example of the first part of a description of the CR IKSWT-19876.

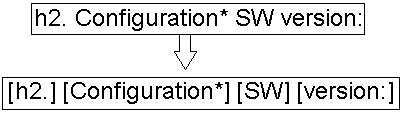


Fig. 5: Example of tokenization process

**3.3.2 Stopwords and punctuation removal**

Once the tokenization process in the CR and TC is completed, it is necessary to remove information that is not relevant to modeling, as punctuation. Additionally, stopwords are words that are irrelevant to the content of a document, such as connectives, articles, pronouns, among others. Thus maintaining more relevant words.

According to what was said before, it was necessary to clean all the data of the CRs and TCs, aiming to obtain a better result of the proposal. To achieve this, a function called “preprocess\_text” was made and it verified if the input was “null” in case it is, the function returns an empty value. In case it is not, the function removed all the special characters.

As an example the setup (Figure 6) of the TC MCA-1011, can be used, as follows:

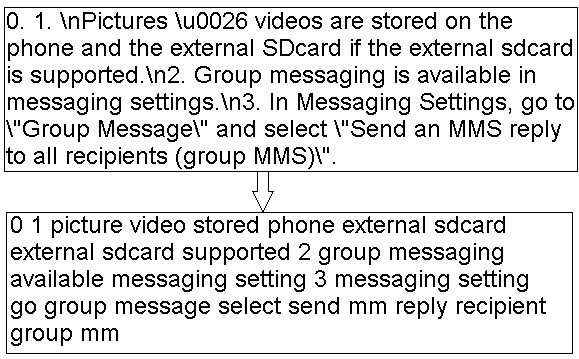


Fig. 6: Removal of stopwords and punctuation

In this example, lemmatization was also carried out, which aims to reduce a word to its base form and group different forms of the same word.

**3.4 Data modeling**

In this phase, a representation of vectors is created, where each dimension is associated with a vocabulary word. In addition, each dimension contains the importance of that word in the text. This importance is calculated with the TD-IDF.CRs and TCs method from the vocabulary of terms extracted in the previous step.

TF-IDF is a weighting technique used in natural language processing to assess the relative importance of a word in a document. It combines the frequency of occurrence of a word in a document (TF) with the frequency of occurrence of the same word in all documents of a corpus (IDF) (Figure 7). The result is a value that indicates the importance of the word in the context of the document and the corpus.

The TF-IDF technique is used in several applications, such as document classification, information retrieval, sentiment analysis and content recommendation. For example, it can be used to identify the most important words in a document or to compare the similarity between two documents [6] as this work proposes. TF-IDF is described by the following equations:

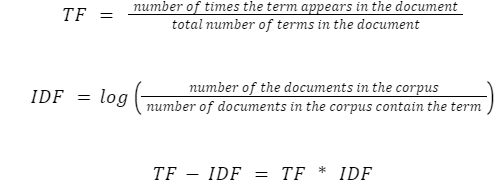


Fig. 7: TF-IDF formula

**3.5 Data analysis**

After the pre-processing and data modeling steps, it is necessary to actually verify the similarity between the TC and the CR. For this, the cosine similarity function was used.

Cosine similarity is a similarity measure used to compare the similarity between two n-dimensional vectors [7]. In natural language processing, it is commonly used to measure the similarity between documents based on their vector representation, e.g., TF-IDF. Cosine similarity calculates the cosine of the angle between two vectors as seen on Figure 8 and gives a value between -1 and 1, where 1 means the vectors are identical and 0 means they are orthogonal.

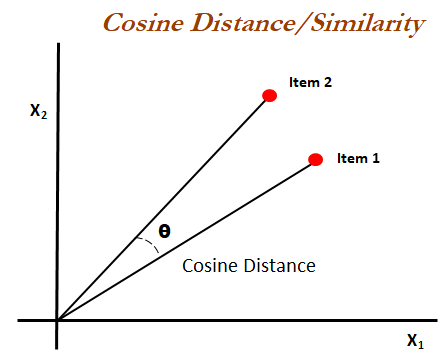


Fig. 8: Similarity between CRs and TCs

**4. RESULTS AND DISCUSSION**

**4.1 Dataset**

In this work, a dataset was created by collecting the CRs WAD which are identified and filled in a spreadsheet by the Common Validation Team, from May 2020 to November 2022, when analyzing the CRs in Android Q, R, S and T. As the spreadsheet already had the ID of the CRs, the Big Query was used to, from the ID of each CR, return the description field of each one, in addition to the ID already provided. The result of this query was saved as a JSON file with 83 samples of CRs.

Also using Big Query, another dataset was created to catch all of the Common Validation Team Test Cases and its information, with 2487 samples, as Figure 9 illustrates.

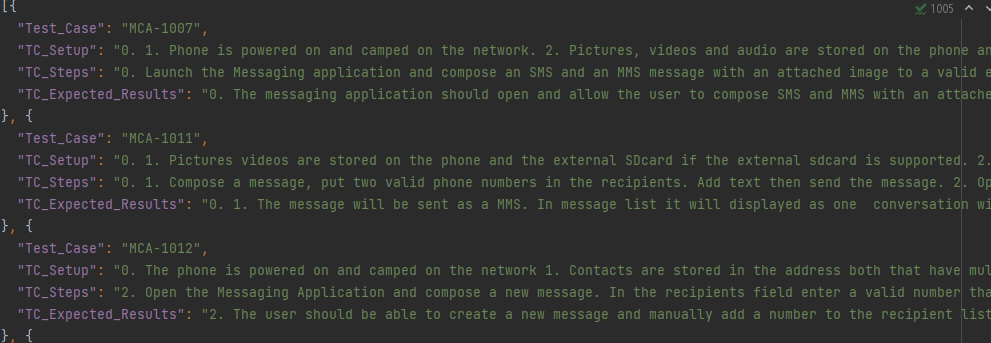


Fig. 9: Part of the JSON used for the TCs

**4.2 Experiments Methodology**

Each CR WAD was given as input to the developed solution and a number of TCs is retrieved, considering the similarity threshold. Each retrieved TC was manually inspected in order to verify whether it requires a refactoring. Then we measure the assertiveness of the results as the percentage of retrieved TCs that actually require factoring. In the experiment, we excluded the TC that was already associated with the CR content. The rationale is to find new TCs that are candidates for refactoring.

In the experiment, the system returned 21 CRs WAD with the threshold defined as 5 at least. But for 18 CRs WAD, the system returned at least two TCs based on similarity, i.e., the TC already associated with the CR (the trivial case) and new candidates for refactoring given the CR. The total number of TCs returned for these 18 CRs WAD is 60 TCs. Of the 18 CRs, 13 of them brought tests that needed to be refactored, that is 72.20% of the valid CRs. And of the 60 TCs, 39 of them needed to be refactored, that is, 65% of the returned TCs..

**5 CONCLUSION**

From the identification of the problem, the search for intelligent solutions began. Thus, it was decided to verify the similarity between the test cases of the team vs the CRs. In this way, if the similarity of the TC received a value considered satisfactory, a manual analysis would then be made and finally define whether the test should be refactored or not.

Finally based on the results presented above, we can see that this work brought a tool to help save time for testers and developers, also in order to help keep the tests updated, bringing better results for the teams, since this brought a result of 71.4% assertiveness. That way we can increase confidence in the TCs because it will be updated more frequently.

To improve the tool in the future, it is possible to conduct an analysis with the ATP team and check if it is possible to use this algorithm with their tool. Additionally, also compare the change requests (CRs) with the results of requirement updates and features and finally, check its assertiveness with different thresholds in order to check its coverage and accuracy.

Another possibility is to check which repository was changed when closing a CR or feature change and check which features and areas are affected and recommend checking all tests related to the repository.

**ACKNOWLEDGMENTS**

I would like to express my gratitude to everyone who has been giving me the support that I needed to complete this work. Firstly I would like to thank my advisor and PO who guided me in this challenge. Finally, thanks to God, my family and friends for all the encouragement.

As a personal thank you, I would like to thank my colleagues, Álvaro Lopes, Ívidy Sousa, Victor Leão, Emerson Sousa and Luiz Pestana, because without their support this work would not have been possible. Finally, a special thanks to Radyja Souza who always encouraged me.

**REFERENCES**

[1] ERLIKH, LEN. Leveraging legacy system dollars for e-business. IT Professional, 2(3):17– 23, 2000.  
[2] BOHNER, S. A. Software change impacts - an evolving perspective. In: Proceedings of the International Conference on Software Maintenance (ICSM’02). Washington, DC, USA: IEEE Computer Society, 2002. p. 263–2. ISBN 0-7695-1819-2.

[3] Mall, R. Fundamentals of software engineering. p.433, 2019.

[4] Batista, G. E. A. P. A. Pré-processamento de dados em aprendizado de máquina supervisionado. Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo. 2003.

[5] Soares, F. A. S. Categorização automática de textos baseada em Mineração de Textos. Ph.D. Dissertação. Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO. 2014.

[6] Christopher D. Manning e Hinrich Schütze. Foundations of Statistical Natural Language Processing, The Mit Press, 1999. p. 218-222.

[7] Manning, C.D., Raghavan, P., & Schütze, H. Introduction to Information Retrieval. Cambridge University Press, 2008. p. 116.