A Source Code Recommender System to Support Newcomers

Yuri Malheiros*§, Alan Moraes†, Cleyton Trindade‡ and Silvio Meira§
*Departamento de Ciências Exatas
Universidade Federal da Paraíba, Rio Tinto, PB, Brazil
Email: yuri@dce.ufpb.br
†Centro de Informática
Universidade Federal da Paraíba, João Pessoa, PB, Brazil
‡ Unidade Acadêmica de Serra Talhada
Universidade Federal Rural de Pernambuco, Serra Talhada, PE, Brazil
§ Centro de Informática
Universidade Federal de Pernambuco, Recife, PE, Brazil

***Abstract*—Newcomers in a software development project often need assistance to complete their first tasks. Then a mentor, an experienced member of the team, usually teaches the newcomers what they need to complete their tasks. But, to allocate an experienced member of a team to teach a newcomer during a long time is neither always possible nor desirable, because the mentor could be more helpful doing more important tasks. During the development the team interacts with a version control system, bug tracking and mailing lists, and all these tools record data creating the project memory. Recommender systems can use the project memory to help newcomers in some tasks answering their questions, thus in some cases the developers do not need a mentor. In this paper we present Mentor, a recommender system to help newcomers to solve change requests. Mentor uses the Prediction by Partial Matching (PPM) algorithm and some heuristics to analyze the change requests, and the version control data, and recommend potentially relevant source code that will help the developer in the change request solution. We did three experiments to compare the PPM algorithm with the Latent Semantic Indexing (LSI). Using PPM we achieved results for recall rate between 37% and 66.8%, and using LSI the results were between 20.3% and 51.6%.**

***Keywords*-recommender systems; software engineering; software maintenance; information theory;**

I. Introduction

Newcomers in a software development project often need assistance to complete their first tasks, because they need to learn how the project works, its architecture, the development process, and how to use some tools to become productive. Then a mentor, an experienced member of the team, usually teaches the newcomers what they need to complete their tasks [1]. To help, the mentor talks to the newcomer, give him tips to solve problems, and usually show source code examples to teach how to do something. However the cost to take an experienced developer to his main tasks to teach a newcomer is high, then sometimes it is not possible to allocate someone as a mentor for a long period of time.

The team interacts with version control system, bug tracking and mailing lists during the development, and all of these tools record artifacts creating the project memory. Recommender systems can use the project memory to help newcomers in some tasks answering their questions, thus in some cases the developers do not need a mentor, since they can ask to the computer.

In this paper we present Mentor, a recommender system to assist newcomers to solve change requests recommending source code files. Mentor uses the Prediction by Partial Matching (PPM)[2] algorithm and some heuristics to analyze a change request and the data of version control systems, and then recommend potentially relevant source code that will help the developer in the change request solution.

We begin the paper with an overview of related work. We then describe in details the Mentor recommender system. We continue by presenting three experiments to evaluate the tool, each one using a different open source project, and their results. We conclude with a discussion of the results, and future research directions.

II. Related Work

The Hipikat [3] assists newcomers in a software development project recommending source code, change requests, mailing list messages, documentation and people information. It creates relations between the artifacts, for example, link source code modified to solve a change request with the change request. This relation is used by Mentor too, but the tools use different approaches to link artifacts. The Hipikat uses the Latent Semantic Indexing (LSI) algorithm to find similarity among textual artifacts. According to the authors, the LSI algorithm is the bottleneck of the system, because it is slow to use LSI in a system with a big number of artifacts. The Mentor uses PPM to find similarity, then we intend to obtain better recommendations than LSI with a good performance, even with many change requests.

Codebook [4] is a framework for mining the data of project repositories. It uses a graph with relations between people and artifacts, an approach very similar to Hipikat.

The Codebook paper presented two applications using the framework: the Hoozizat, a tool to find experts and the Deep Intellisense, a Visual Studio add-in that shows events ordered chronologically related to a symbol. The Codebook could be used to recommend source code related to a change request, if the graph has edges linking these kinds of artifacts.

The system proposed by Moin and Khansari [5] also recommends source code related to change requests. The tool uses the Support Vector Machine (SVM) classifier [6] to find similar change requests solved earlier and then recommend the source code modified to solve these similar change requests as the solution of an open change request. The system has a crucial difference to the Mentor, because it recommends whole directories with source code files, and Mentor recommends files directly. To recommend directories may not help the developers, because the directories may have a lot of files.

III. Mentor

Mentor is a tool that manages change requests and makes recommendations of source code related to a change request.

The tool makes recommendations assuming similar change requests have similar solutions. Thus, to find the source code files related to an open change request, the tool looks for similar change requests that were solved in the past and recommends the files changed to solve them as the related files of the open change request. The tool ranks the solutions by the similarity of the change requests, then the files modified to solve the most similar change request appears first, the files of the second most similar change request appears next, and so on.

The Mentor recommendations are independent of programming language and independent of the language used to describe the change requests. Then, it does not matter if the project is written in C, C++, Java or mix programming languages, or if the developers using English or Portuguese to describe the change requests, the algorithms used in the tool work in all these cases. This is a very good point of Mentor, because different teams, developing different kinds of projects, can use the same tool and obtain useful results.

The tool is based in the MVC (Model-View-Controller) architecture and two components were created to work on the recommendation tasks - Matcher and Similarity Assigner - they interact directly with the Model layer reading and writing data in the database.

A. Similarity Assigner

The Similarity Assigner component creates the similarity relations among change requests. It analyzes each change request stored in the database, and it uses the PPM algorithm to calculate similarity between them. The process to identify similar change requests has two steps: model creation and classification.

Context k=1			
h	o	1	1/2
	escape	1	1/2
o	c	2	2/3
	escape	1	1/3
c	u	2	2/3
	escape	1	1/3
u	s	2	2/3
	escape	1	1/3
s	p	1	1/2
	escape	1	1/2
p	o	1	1/2
	escape	1	1/2
Context k=0			
	h	1	1/16
	o	2	2/16
	c	2	2/16
	u	2	2/16
	s	2	2/16
	p	1	1/16
	escape	6	6/16

Table I
PPM model after process the string "hocuspocus".

In the first step, a PPM model is created for each change request stored in the system. It is made using the text of the change request summary, description and comments made by developers concatenated in only one string. The PPM model is a statistical model, i.e., the model is made by probabilities according to the occurrence of the text symbols.

The model also considers the context of the symbols, i.e., the k previous symbols of the current symbol. Using contexts, the probability of a symbol does not just depend on its frequency, but it depends on the context in which it occurs too. For example, the probability of the letter "h" appears in an English text is 5%. However, if the current symbol is the letter "t", there is a greater probability that the next symbol is the letter "h", about 30%, because, in English, the letters "th" often appear together [2].

There is a special symbol called escape in the PPM algorithm, it is added to the model in a context whenever a symbol appears for the first time in that context. This special symbol is important to calculate the entropy of a message, because it represents the probabilities of all the symbols that do not appear in the model.

The Table I shows a PPM model using a maximum context $K = 1$ for the string "hocuspocus". N is the number of times a symbol appears in a context and P is the estimated probability for this symbol.

We use the entropy to measure how a change request is similar to other. The entropy is calculated using the PPM model. Thus, to calculate the similarity between a change request A and a change request B, we need to create a PPM model of A and then calculate the entropy of B using the model of A. The result will give to us if they are similar or not.

Mentor uses two equations to find the entropy. The Equation (1) is the mathematical definition for the information I of a symbol x [7].

The entropy H of a text is the mean of the information produced by the symbols of this text. The Equation (2) represents the averaging of information of each symbol x_i of a message of size N. It is easy to notice that the value of the entropy depends directly of the probabilities of the model used.

The tool calculates the information of each symbol of the summary, description and comments concatenated of the change request B using the probabilities of the change request A model, sum them all and divide by the message length.

A low entropy value means that the change requests are similar, a high value means the opposite, that the change requests are not similar.

$$[h]I(x) = \log_2(\frac{1}{P(x)}) \tag{1}$$

$$[h]H = \frac{1}{N}\sum_{i=1}^{N} I(x_i) \tag{2}$$

B. Matcher

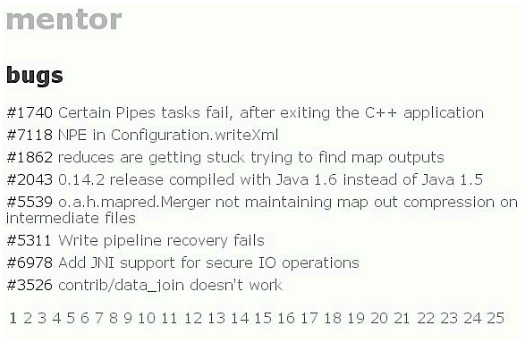
The Matcher component analyzes every change requests stored in the database and the data of the version control used in the project, and it uses a heuristic to discover and store the relation between the revisions and change requests in the database.

The heuristic used by the Matcher works as follows. Usually in software development project the developers use a convention to create commit messages. When a developer sends the modifications to solve a change request, he could attach a message like "issue #1234 solved", where #1234 is the ID of the change request solved. Thus, the Matcher uses regular expressions to scan the commit messages looking for some patterns like these. When it finds a pattern, a relation between a change request and a revision is stored in the database.

C. Usage

The Figure 1 shows the Mentor initial screen. The tool displays a list of the change requests IDs and its summaries ordered by modification date. This approach is very common in bug tracking systems, because the users can browse quickly among the change requests and read the short description to know what the change request subject is.

Clicking on a change request, the Mentor change its screen to the change request details, it is the screen of the


mentor

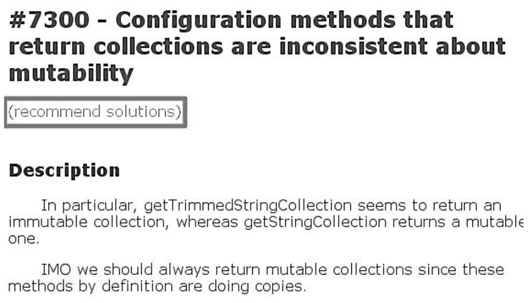
bugs

#1740 Certain Pipes tasks fail, after exiting the C++ application
#7118 NPE in Configuration.writeXml
#1862 reduces are getting stuck trying to find map outputs
#2043 0.14.2 release compiled with Java 1.6 instead of Java 1.5
#5539 o.a.h.mapred.Merger not maintaining map out compression on intermediate files
#5311 Write pipeline recovery fails
#6978 Add JNI support for secure IO operations
#3526 contrib/data_join doesn't work

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Figure 1. Mentor index

Figure 2. In this screen the developer can analyze the change requests details, in many cases only the information of the summaries is not sufficient to inform the developer what he needs to start to solve the problem.


#7300 - Configuration methods that return collections are inconsistent about mutability

(recommend solutions)

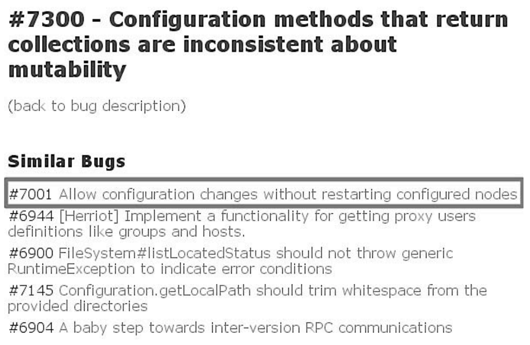
Description

In particular, getTrimmedStringCollection seems to return an immutable collection, whereas getStringCollection returns a mutable one.

IMO we should always return mutable collections since these methods by definition are doing copies.

Figure 2. Change request details

Below the change request summary there is a link highlighted with the text "recommend solutions". Clicking on this link, the Mentor will recommend to the developer a list of similar change requests that were solved in the past. The figure 3 shows the similar change requests of the request #7300 of the project Hadoop Common.


#7300 - Configuration methods that return collections are inconsistent about mutability

(back to bug description)

Similar Bugs

#7001 Allow configuration changes without restarting configured nodes
#6944 [Herriot] Implement a functionality for getting proxy users definitions like groups and hosts.
#6900 FileSystem#listLocatedStatus should not throw generic RuntimeException to indicate error conditions
#7145 Configuration.getLocalPath should trim whitespace from the provided directories
#6904 A baby step towards inter-version RPC communications

Figure 3. Similar change requests

The most similar change request according to the tool is the change request #7001. Clicking on the change request in the list, the Mentor shows the revisions related to the change request and the source code files changed in these revisions.

IV. Experiment

We compared PPM, the technique used in Mentor, with LSI, the technique used in Hipikat [3] to evaluate which technique is more efficient to find similarity among texts.

We create other version of the Similarity Assigner component to use the LSI instead PPM, thus, we change only the algorithm to find similarity, the rest of the system remained exactly the same. We use the LSI implementation of the Gensim library[1] to generate the text similarity relations.

A. Metrics

We used three following metrics in the experiment: precision[8], recall[8] and recall rate[9], because they are largely used in the literature in similar experiments.

B. Hypotheses

For the experiments the null hypothesis are:

- $\mathbf{H}_{n1}$ - PPM *precision* = LSI *precision*
- $\mathbf{H}_{n2}$ - PPM *recall* = LSI *recall*
- $\mathbf{H}_{n3}$ - PPM *recall rate* = LSI *recall rate*

And the alternative hypothesis are:

- $\mathbf{H}_{a1}$ - PPM *precision* $>$ LSI *precision*
- $\mathbf{H}_{a2}$ - PPM *recall* $>$ LSI *recall*
- $\mathbf{H}_{a3}$ - PPM *recall rate* $>$ LSI *recall rate*

C. Instrumentation

We did three experiments, each one using data of a different open source project. The projects used are:

- **GTK+**: 503,161 source code lines in 1,348 files;
- **GIMP**: 737,835 source code lines in 3,293 files;
- **Hadoop**: 613,481 source code lines in 1,003 files.

We filter some change requests in the experiment. First, we selected only the minor and trivial change requests, because they are simple tasks that a newcomer could solve.

Some change requests imported do not have any version control revision associated, because there is no revision referencing them in the commit messages. These requests were removed from the experiment too.

Also, the solution of some change requests may does not have intersection with any other change request solution. This is a problem, because it is impossible to recommend a correct solution if there is not any change request that changes the correct files. For example, if a change request (A) was solved modifying a file (X), and in all the rest of the change requests, no one was solved modifying the file (X), it is impossible to recommend the correct solution of change request (A). The requests like the request (A) were removed from the experiment too.

The Table II shows the amount of change requests and revisions used in the experiment, and the period of time.

[1]http://nlp.fi.muni.cz/projekty/gensim/

Project	Data	Amount	Period
GTK+	change requests revisions	374 26805	06/2001 - 08/2008
GIMP	change requests revisions	496 29708	06/2001 - 09/2008
Hadoop	change requests revisions	250 3049	06/2009 - 06/2011

Table II
Experiment data

D. Validities

All the variables in the experiments are static, we change only the similarity algorithm to compare PPM and LSI. In other words, in the experiments we change the similarity algorithm applied in the static variables and observe the metrics.

To reject or not the hypotheses we use two statistical tests, the Wilcoxon signed-rank test for matched pairs for precision and recall, and a proportion test for the recall rate [10].

We used three different projects to avoid mistakes in the experiment conclusion. If we used only one project, maybe the results were true only in this specific case, then we would not be able to conclude if the PPM is better or not compared to LSI. However, using three projects, written in different programming languages and with purposes very distinct, we try to avoid this threat.

E. Execution

For each experiment, we import the project data using some scripts to load the data from different formats directly in the database of the system and run the processes to generate similarity and change requests-revision relations, one time with PPM and other time with LSI.

Each change request received from one to ten recommendations of similar change requests, and we calculated precision, recall for each recommendation. For example, for one recommendation (A), first only one change request, and the files changed to solve it, is recommended, then the metrics are calculated. After that, the system recommends other change request, the second most similar. Now, the metrics are calculated using the files of the first and of the second change request. This process continues until recommends ten change requests and their files. In the end, we calculated recall rate for set of recommendations.

All change requests used in the experiment were solved previously by the developers of the projects, however, we did not use this information during the recommendation process. Thus, to analyze if a recommendation is right or wrong, we only need to compare the files recommended by the system with the files of the real solution.

The calculation of the metrics precision and recall is realized for each change request, then, for example, recommending one similar change request we get many values of precision and recall, one for each change request that receives recommendation. The same happens to two, three, four similar change requests recommendations, and so on. The results presented for the precision and recall in the tables are the means of the results of all change requests. We use this approach to simplify the data presentation.

The tables III, IV and V show the results of the three experiments. The first column shows the number of similar change requests recommended, and the next columns show the mean of the metrics precision and recall, and recall rate respectively.

PPM			
N	precision	recall	recall rate
1	17.8476%	18.6704%	22.1925%
5	7.7356%	37.7189%	44.1176%
10	4.9950%	46.5235%	53.7433%
LSI			
N	precision	recall	recall rate
1	2.6744%	1.7342%	3.2086%
5	1.7887%	8.0765%	10.6952%
10	1.1780%	15.9488%	20.3209%

Table III
GTK+ EXPERIMENT RESULTS

PPM			
N	precision	recall	recall rate
1	11.3363%	11.5758%	21.2%
5	5.987%	32.9433%	53.2%
10	4.2510%	45.9387%	66.8%
LSI			
N	precision	recall	recall rate
1	6.9959%	7.9908%	15.6%
5	3.3429%	19.9785%	36%
10	2.5328%	32.7126%	51.6%

Table IV
HADOOP EXPERIMENT RESULTS

V. DISCUSSION

All results using PPM to find similar change requests were better than the results using LSI. Let's analyze the hypotheses always using the case that we recommend ten change requests.

The experiment using GTK+ had 4.9950% of precision using PPM and 1.1780% using LSI. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 6640$, $z = -4.9494$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n1}. For recall in the GTK+ project, the PPM had 46.5235% and LSI had 15.9488%. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 3096$, $z = -5.9436$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n2}. For the recall rate in GTK+ project, the PPM had 53.7433% and LSI had 20.3209%. The proportion test with 0.05 of significance returns $P - Value = 0$, $z = 9.4648$ and the critical z is +-1.96, then, we can reject the null hypothesis H_{n3}.

The experiment using Hadoop had 4.2510% of precision using PPM and 2.5328% using LSI. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 11393$, $z = 4.4732$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n1}. For recall in the Hadoop project, the PPM had 45.9387% and LSI had 32.7126%. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 11717$, $z = 22.0381$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n2}. For the recall rate in Hadoop project, the PPM had 66.8% and LSI had 51.6%. The proportion test with 0.05 of significance returns $P - Value = 0.0005$, $z = 3.4579$ and the critical z is +-1.96, then, we can reject the null hypothesis H_{n3}.

The experiment using GIMP had 2.6758% of precision using PPM and 1.8808% using LSI. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 27454$, $z = 15.9605$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n1}. For recall in the GIMP project, the PPM had 24.6009% and LSI had 18.5938%. The Wilcoxon signed-rank test for matched pairs with 0.05 of significance returns $T = 13375$, $z = 14.4705$ and the critical z is +-1.959962, then, we can reject the null hypothesis H_{n2}. For the recall rate in GIMP project, the PPM had 37.0968% and LSI had 29.4355%. The proportion test with 0.05 of significance returns $P - Value = 0.0104$, $z = 2.5607$ and the critical z is +-1.96, then, we can reject

PPM			
N	precision	recall	recall rate
1	7.5013%	7.1029%	13.1048%
5	3.7490%	16.87%	28.0242%
10	2.6758%	24.6009%	37.0968%
LSI			
N	precision	recall	recall rate
1	5,4571%	5.6412%	8.871%
5	3.0886%	13.9663%	22.1774%
10	1.8808%	18.5938%	29.4355%

Table V
GIMP EXPERIMENT RESULTS

the null hypothesis H_{n3}.

The recall rate is a very important metric in these experiments, because the Mentor may make recommendations of change requests that were solved modifying some correct files, but also other files. This recommendation still be useful for the developer, however, the precision and recall values tend to low.

For example, the change request #7300 of Hadoop Common project was solved changing three .java files (Configuration.java, TestConfiguration.java and StringUtils.java). Using PPM, the most similar change request recommended by Mentor was #7001, and it was solved changing seven .java files, including Configuration.java. In this case, the change request enters as a correct answer in the recall rate calculation, but the precision was only 14.2%. Despite the low value of precision, the recommendation of the seven .java files may be very good, because a developer is likely to open the TestConfiguration.java file when he finds the Configuration.java file, for the reason that the first is the test file of the second. Further, in the Configuration.java there are many uses of methods of the class StringUtils, then, a developer reading the code will open the StringUtils file too. Using LSI the change request #7300 appeared as the 8th most similar change request.

The Hadoop project follows a rigid commit message guideline, the relation between a revision and a change request is very clear. The GIMP project, that had low results, does not follow clear rules to create commit messages. However, the GTK+ project that had as good results as Hadoop does not follow any clear rule too. So, the rigid commit messages pattern may do not have so much influence, because the Matcher does a good job looking for the change requests ID.

After the three experiments we can conclude that to use PPM to get similarity among text is potentially better than to use LSI.

VI. Conclusion

To move an experienced developer from his main tasks, to act as a mentor may cause a delay in the project and increase its cost. In this paper we presented a tool called Mentor that tries to help the developers to solve change requests, and to avoid moving experienced members from their tasks.

We ran three experiments to evaluate the tool comparing PPM with LSI to find similarity and all null hypotheses were rejected in the three experiments. Using PPM we achieved results for recall rate between 37% and 66.8%, and using LSI the results were between 20.3% and 51.6%.

There are many possibilities for future work. The first is expanding the recommendations to other artifacts, for example, messages of a mailing list or the documentation of the project. They are made by text, then it seems possible to use PPM to find similarity among all these artifacts.

The recommendation engine of Mentor could be integrated in some project management systems. This way the project does not need to change de tool to manage the project and the Mentor will only add some important features related to recommendation in the bug tracking system of the management tool.

Other important work we need to do is evaluate the Mentor using a case study. Developers using the tool could show to us if the tool is really useful in real tasks of real software projects.

Acknowledgment

The authors would like to thank the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq) and the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) for funding this research.

References

[1] C. Hansman, V. Mott, A. Ellinger, and T. Guy, *Critical Perspectives on Mentoring: Trends and Issues*. Columbus, OH: Clearinghouse on Adult, Career and Vocational Education, 2002.

[2] D. Salomon and G. Motta, *Handbook of Data Compression*, 5th ed. Springer Publishing Company, Incorporated, 2009.

[3] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, “Hipikat: A project memory for software development,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 446–465, 2005.

[4] A. Begel, Y. P. Khoo, and T. Zimmermann, “Codebook: discovering and exploiting relationships in software repositories,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE ’10. New York, NY, USA: ACM, 2010, pp. 125–134.

[5] A. H. Moin and M. Khansari, “Bug localization using revision log analysis and open bug repository text categorization,” in *OSS*, 2010, pp. 188–199.

[6] V. Vapnik, *Statistical learning theory*. Wiley, 1998.

[7] C. E. Shannon, “A mathematical theory of communication,” *The Bell system technical journal*, vol. 27, pp. 379–423, Jul. 1948.

[8] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[9] P. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 499–510. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.32

[10] M. F. Triola, *Introdução à Estatística, 10a. edição*. LTC, 2008.