A Corpus Builder: Retrieving Raw Data from GitHub for Knowledge Reuse In Requirements Elicitation

Roxana Lisette Quintanilla Portugal, Hugo Roque, Julio Cesar Sampaio do Prado Leite
Departamento de Informtica, PUC-Rio / Rio de Janeiro RJ 22451-9000, Brasil
{rportugal, julio}@inf.puc-rio.br
hugo.roque@aluno.puc-rio.br

Abstract

Requirement elicitation is an important task, which can lead to cost reduction in the overall software process, as it avoids failures due to lack of proper understanding about what to build. However, usually, there is a lack of time devoted to proper elicitation during software construction. We assume information from similar projects is a valuable knowledge for requirements engineers when facing a new project in the same or related domain, and its acquisition can be speeded up by knowing their main features. This information is usually located in Readme documents of GitHub. We present a tool that helps in handle this large amount of information by retrieving a corpus of Readme documents given a domain-related query. It is described, in detail, how a corpus is created and stresses the importance of having a quality corpus as base for data mining, or as input for tools of qualitative data analysis.

1 Introduction

Imagine the following situation: a group of musicians is looking to produce a music application; they believe it could be a hit. They contacted angel investors, which are willing to invest, but needed more details about the idea. As such, they decided to hire a requirements engineering company to organize the intentions, before contracting a software developer company to build the application. The musicians overall idea is to have an application user to know a city, a neighborhood, or a place like a university, by the music that it is being listened around.

It happens that the requirements engineering company, hired to do the job, is not familiar with the domain and would have to quickly gain leverage on the contextual knowledge to better collaborate with the musicians, as well as to build a proper requirements for future developers. This contextual knowledge must be both related to the client side, but also to the possible software ecology where the application will operate.

We depart from the assumption that requirements-related information can be elicited from Big Data, in this case we use the software repository GitHub, since this source owns, to date, more than 35 million of projects in its repository (Metz, 2016). This assumption is founded on the evidence that projects on GitHub has encoded knowledge. Although this encoded knowledge is mainly represented in programming languages, there are annotations in natural language that describe the project purpose. Of course, those projects are different in several manners, either in quality of its contents, as well as in the level of information provided in natural language. However, most of the projects we have retrieved from this repository do provide some natural language texts i.e., the Readme document of each project, which helps in understanding a project purposes. Our work is contextualized in what (Markus, 2001) calls secondary knowledge miner which is defined as “people who seek to answer new questions or create new knowledge through analysis of records produced by other people for different purposes” and “extract knowledge from records that were collected by others, possibly unknown to the reusers…”. Markus also noted that this reuse is not limited to structured data. “Although most research on data mining has focused on structured data, this is data on databases or knowledge datasets, similar issues are likely to apply in the case of secondary reusers of documents”.

A requirements elicitor could perform a manual revision of GitHub projects given a domain-related query; however, the reading of hundreds of projects may not be efficient in time-constrained settings. For instance, a work from EMSE[1] field mentions that researchers manually extracted data from 32 publications published in digital libraries which it took 80 hours for two tasks: (1) extraction and (2) analysis of data (Ekaputra et al., 2014). Nowadays both Digital Libraries and GitHub own a plethora of data; on that ground we automate the documents extraction task from projects hosted on GitHub, this time using its Readme perspective. Thus, a set of documents is ready for the analysis task that can be performed manually assisted by tools for qualitative data analysis. e.g., Atlas.ti or NVivo. Or to perform an automatic analysis by using text-mining techniques.

The remainder of this paper is structured as follows. Section 2 provides a research baseline for motivation. Section 3 explains the rationale for selecting artifacts in GitHub. Section 4 details the design and construction of the tool. Section 5 describes the qualitative analysis conducted in the corpus of Readmes for the domain "music application". Section 6 concludes and points out future work.

2 Corpus of Documents

(Sinclair, 2005) states this principle when building a Corpus: "The contents of a corpus should be selected regardless of their language, but according to their communicative function in the community in which they occur". In this respect, a previous work (Portugal et al., 2015) performed an exploratory research to verify to what extent is feasible the Readme document for use in requirements engineering. In this regard, the Readme perspective of GitHub projects has the communicative function to describe the main features of a project. Similarly, the Issues perspective has the communicative function of tracking the evolution of software features.

Another aspect about the construction of a Corpus is that it can be considered as the first step towards the building of a web extraction tool, specifically a Natural Language Processing NLP-based wrapper (Laender et al., 2002). A Similar approach to extract data given a query can be found in the tool Webcorp (Renouf, 2003) however, this tool does not cope with our goal, as they mainly use the Google API for gathering information; as such this mechanism does not cover the internal documents of GitHub projects. Another project, very similar to ours is the GHTorent (Gousios, 2013), which in fact, can accomplish more than a retrieval of Readmes, making database dumps of all of the projects on GitHub. However, we found some technical barriers, specially for users not used to deal with this kind of technology. For instance, to get projects related to a query, the user may need to download a database dump (around 30GB size in 10 hours) and then supported by a Database Management System (DMBS) as MySQL, the user will be able to query in SQL format the information needed. Instead, we are proposing a service to deal with other type of queries; thinking in requirements, our specification would be:

> Given a query e.g "music application" the user may be able to download a zip file of Readmes with extension .txt, numbered by order of result's appearance, and each document should be named with the project and its owner name.

Another barrier to accomplish our goal with (Gousios, 2013) work is that the schema of GHTorent database dump[2] does not contain a table related to Readmes information. However this this tool would be useful when retrieving GitHub Issues.

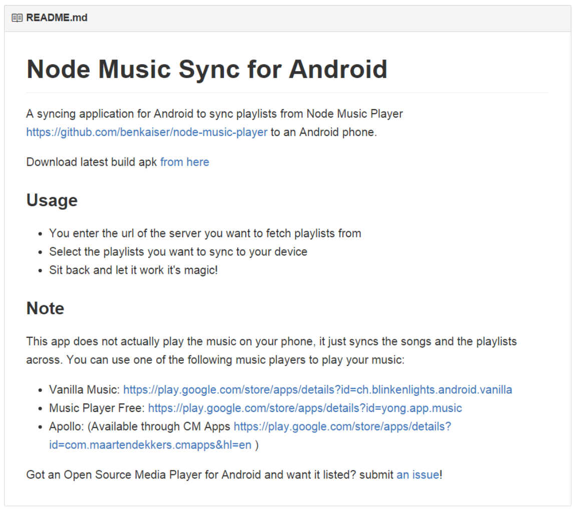
Another constraint that motivated us to build a service, is that Readmes can vary daily on GitHub, when new projects are created or when the existing change its relevant. This relevance is given by the Forks, Stars, Pull-request, number of Issues and Comments a project receives.

3 GitHub Perspectives for Requirements Elicitation

Using the concepts of viewpoints and perspectives (do Prado Leite and Freeman, 1991) we, as requirements engineers, see the GitHub in the following way: A project, specifically an application, can express a viewpoint, i.e. a way to address what the user needs in certain domain. It happens that each project, viewpoint, may use several representations (perspectives), to describe a project. These GitHub perspectives are: Readmes, Issues, Issue's

[1]Empirical Software Engineering

[2]http://ghtorrent.org/dblite/


README.md

Node Music Sync for Android

A syncing application for Android to sync playlists from Node Music Player https://github.com/benkaiser/node-music-player to an Android phone.

Download latest build apk from here

Usage

- You enter the url of the server you want to fetch playlists from
- Select the playlists you want to sync to your device
- Sit back and let it work it's magic!

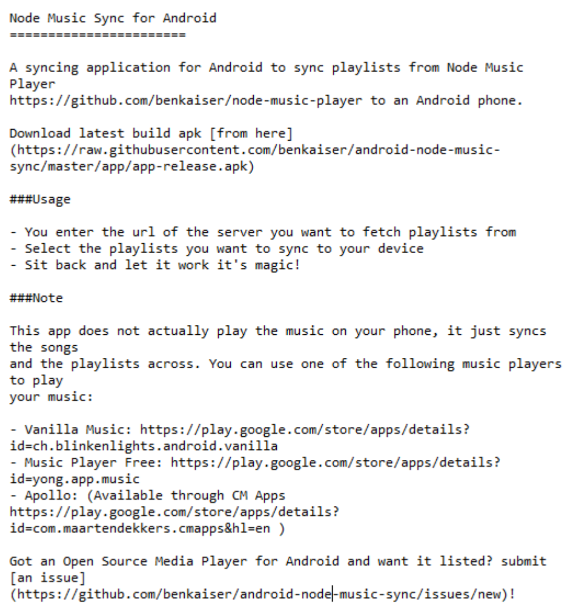
Note

This app does not actually play the music on your phone, it just syncs the songs and the playlists across. You can use one of the following music players to play your music:

- Vanilla Music: https://play.google.com/store/apps/details?id=ch.blinkenlights.android.vanilla
- Music Player Free: https://play.google.com/store/apps/details?id=yong.app.music
- Apollo: (Available through CM Apps https://play.google.com/store/apps/details?id=com.maartendekkers.cmapps&hl=en)

Got an Open Source Media Player for Android and want it listed? submit an issue!

Figure 1: A Readme on GitHub


```
Node Music Sync for Android
=======================

A syncing application for Android to sync playlists from Node Music
Player
https://github.com/benkaiser/node-music-player to an Android phone.

Download latest build apk [from here]
(https://raw.githubusercontent.com/benkaiser/android-node-music-
sync/master/app/app-release.apk)

###Usage

- You enter the url of the server you want to fetch playlists from
- Select the playlists you want to sync to your device
- Sit back and let it work it's magic!

###Note

This app does not actually play the music on your phone, it just syncs
the songs
and the playlists across. You can use one of the following music players
to play
your music:

- Vanilla Music: https://play.google.com/store/apps/details?
id=ch.blinkenlights.android.vanilla
- Music Player Free: https://play.google.com/store/apps/details?
id=yong.app.music
- Apollo: (Available through CM Apps
https://play.google.com/store/apps/details?
id=com.maartendekkers.cmapps&hl=en )

Got an Open Source Media Player for Android and want it listed? submit
[an issue]
(https://github.com/benkaiser/android-node-music-sync/issues/new)!
```

Figure 2: Raw Readme data

Comments, Commits, Commit's Comments, and Gits. We argue that each of this artifacts express a perspective of a particular viewpoint (project) because, on the Readme perspective a user is able to see a summary of features that the application implements. In the Issue perspective it is possible to get more specialized information about features (e.g. bugs or enhancements). Even more, it is possible to see the decisions (Comments perspective) taken about an issue before been implemented.

3.1 The Readme Artifact

(Kupiec et al., 1995) state that "Abstracts are sometimes used as full document surrogates, for example as an input to text search systems, but they also speed access by providing an easily digested intermediate point between a documents title and its full text, that is useful for rapid relevance assessment". We judge that Readmes have the role of abstracts on GitHub environment. Fig.1 shows the Readme document of the project **android-node-music-sync** from user **benkaiser**. This project was found with the query "music application android". Using the GitHub API v3.0 to access the data, we obtained its raw version Fig.2. As we analyzed the raw data, we figure it out that in our ongoing research, we will be facing the mining of documents of different nature, this is, structured data: source code, semi-structured data: documents with markups such us html, xml, markdown[3] among others, and unstructured data: free texts in comments and other documents. This time, by using the Readme document we are dealing with semi-structured texts, as most of the Readmes follow the predefined markdown format. For instance (see Fig.2) to indicate an url, they used [texto](url). In other exemplars we found ![alt text](image path) to indicate an image.

3.2 The Requirement-Related Information

What we pursue with a corpus of Readmes is the finding requirement-related information, which are phrases that can be mined to give an idea of the project purposes. Thus, the reader can reuse this knowledge for learning or generating new ideas in requirements elicitation tasks. From Readme (Fig.1) some candidate phrases to be mined would be:

"A syncing application for Android to sync playlist from Node Music Player to an Android phone"
"This app does not actually play the music on your phone, it just syncs the songs and the playlist across. You can use one of the following music players to play your music"

4 Working Towards the Tool

As we started to explore GitHub, we built a script to extract readmes just for the query "Real Estate in: readme" (Portugal et al., 2015). This serve us for our initial purpose of discover ideas and find *domain-independent regularities* (Arora et al.,

[3]Markdown is a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML. source: Wikipedia

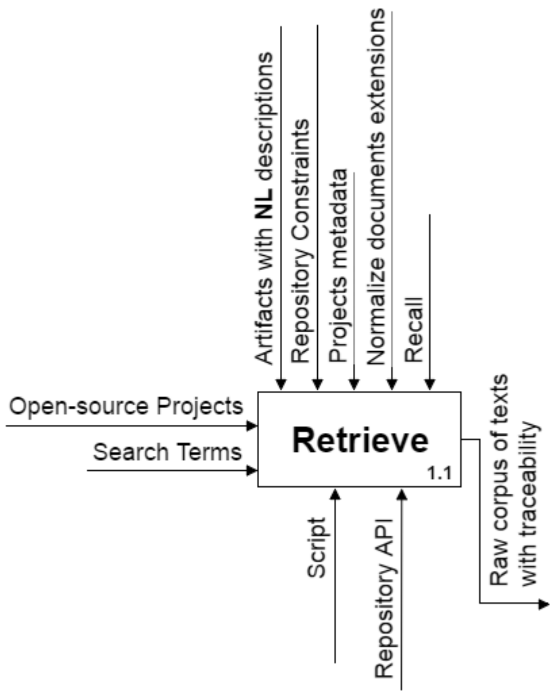


Figure 3: SADT Model for Retrieval Process

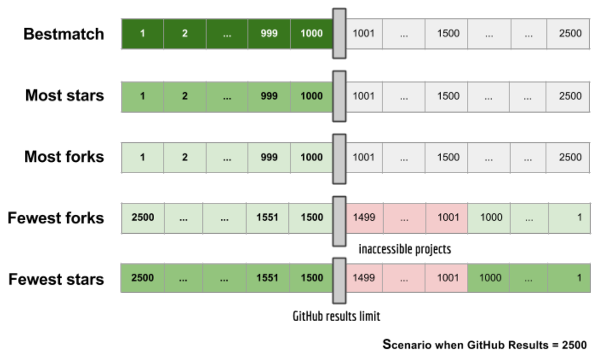
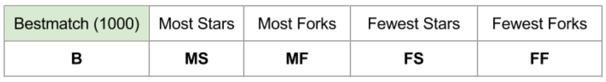


Figure 4: Querying by combining sorting options



Bestmatch (1000)	Most Stars	Most Forks	Fewest Stars	Fewest Forks
B	MS	MF	FS	FF

Figure 5: Organizing GitHub results

2014), (Ridao et al., 2001) which may allow us to find requirement-related information. Following, using the SADT (Structured Analysis and Design Technique) (Chen, 1976) we modeled a process to address our approach. Fig.3 highlights one of the activities, Retrieve, which points the construction of this tool.

4.1 The Retrieve Activity

The Retrieve describes the **inputs**: the domain-related query (search terms) and the GitHub open-source projects. With this, it is requested the projects that match with the query. The **constraints**: our process was designed to be suitable for any artifact with natural language descriptions. It was considered the request limits using the GitHub API[4]. We took care in backward traceability; thus, once a Readme is in a corpus it is possible to locate its source on GitHub. A concern is the quantity of search results limited to 1000, this fact, made us to think in a situation where the project 1001 could be the interesting one for an requirements elicitor; therefore, we created heuristics taking advantage of GitHub metadata to improve the recall of results. Finally, we had to deal with a variety of document extensions (.md, .rtf, .html, .doc, etc.) and normalize them to .txt before a Readme is inserted in the corpus. The unique situation where a Readme is not retrieved, is when this is located out of the root of its project. The **outputs**: it is expected the corpus of Readmes in .txt format and the package zipped containing them.

[4]GitHub API v3: Rate Limiting. https://developer.github.com/v3/#rate-limiting

4.2 Heuristics

In order to bring more than 1000 results through the search, our tool explored five of the possible sorts a user may do: **best match, most starts, fewest stars, most forks, fewest forks**. Each sorting becomes a new query. We combine them to surpass the 1000 results limitation (Fig.4), and with the current GitHub API we were able to perform those five queries in a single task.

There is a possibility of leaving out many projects (see the cells in gray and red), that is because, it is shown just the first 1000 results of any sorting query; after that is not guaranteed the order of projects relevance. We had another concern, which is the project rating given by users, giving a **star** or performing a **fork**, resulting in projects repeated in any of the sorting operations. Our heuristic uses a union operation in order to capture those intersections. Finally, we organize the corpus in the order shown in Fig.5. As a user would not be able to get this extra through GitHub website, we consider we improved the recall of results.

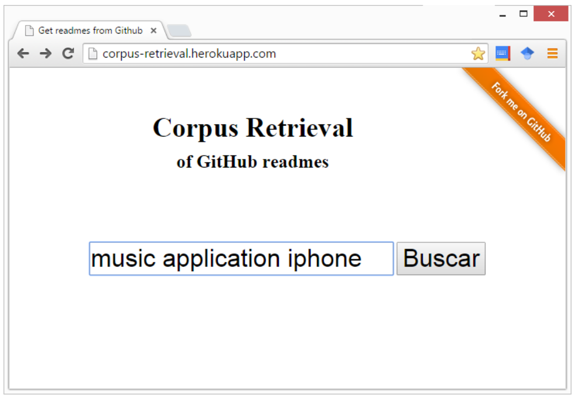


Figure 6: Web application for corpus retrieval

4.3 Tool Presentation

The tool[5] presentation (Fig.6) is simple and just for the purpose to retrieve a corpus of Readmes given a query. To continue with the GitHub spirit providing Open-source software (OSS) to the community, we made available the script[6] code and also a complementary script[7] developed to support the performance of the application on web browsers.

5 Analyzing the Raw Data Retrieved

Our assumption that a corpus of Readmes could be useful for finding requirements-related information is based on knowledge reuse literature as well as on our own evidence. This premise can be questionable, as the Readme is a brief user documentation of software projects and may seen hardly a reliable way for obtaining requirements for your own project. In fact, The best one can discover is what features these other projects offer for later reuse. The usefulness of Readme documents is the identification of relevant projects, for later exploration of another perspectives (Issues, Commits) which probably does contain more data with the stereotypical of requirements. In this regard, we are looking for the type of reuse that (Goldin and Berry, 2015) state "Reuse can take place during any phase of a computed-based system development, including during proposal consideration and marketing analysis, requirements elicitation, requirements analysis, architecture design, code implementation, and testing. . . Thus, reusing requirements can be most beneficial, because if it leads to off-the-shelf reuse of the required product, resulting in greatest reduction of development effort and time to market".

We wanted to test two hypotheses:

Hypothesis 1: A corpus builder of Readmes permits the finding of features using a similar-based projects approach.

Hypothesis 2: The mined requirements-related-information is useful for reuse.

For this, we built a corpus for the "music application" query with 1206 Readmes and took a representative sample of 291 Readmes to be read manually with the aim to find reusable knowledge. Once some *phrases in context* with high chances of being re-used are identified, they were shown for a Music Aplication Startup.

It is worth noting that the selection of Readmes was conducted randomly with an script[8] we created for future test.

5.1 Findings

A remarkable finding from our notes is the GitHub limitation[9] which leads to not support a phrase query, resulting in Readmes vaguely related to the "music application" query. This happens because some Readmes contained just the word "music" and others only with"application". This fact impacts the precision to filter relevant projects within a corpus created.

As we are investigating patterns to anchor requirements, the manual reading allowed us to see some patterns motivated by the work of (Arora et al., 2014). We identified six concurrent patterns (Table 1) and then we mined them on the entire corpus (2016 Readmes) to obtain its frequency of appearance.

To answer **hypothesis 1 and 2**, we select the pattern with the lowest rank "allows users to". We grouped 861 Readmes with similar file size (0kb-1kb), and the mining of this pattern using the package Qdap (Goodrich et al., 2016) for R project, resulted in 14 Readmes matching within these Readmes, then a manual extraction of phrases in context was done. Below we show four phrases shown to a music application startup[10], and in ***italic***, the startup assessment.

[5]http://corpus-retrieval.herokuapp.com/

[6]https://github.com/nitanilla/corpus-retrieval

[7]https://github.com/nitanilla/github-proxy

[8]https://github.com/nitanilla/Random-Readme

[9]https://help.github.com/articles/searching-code/#considerations-for-code-search

[10]Hear: https://www.facebook.com/apphear

Table 1: Requirement Patterns in Readmes

Requirement Pattern	In # of Readmes
`to provide`	77
`which can`	57
`can be used to`	43
`should be able to`	38
`that allows you to`	31
`allows users to`	28

1.- Moment is a web application that stores all your special memories with the music you love. By tapping into Spotify's Web API, Moment allows users to bookmark music they enjoy in a journal format and navigate all their previous memories with music. ***A good idea to take into consideration***

2.- SoundShack Leverages the Power of Broadcom's Latest Wiced chip to Stream HD Audio to Wifi enabled android devices. This Android application allows users to syn-chronously stream music with no lag. User can also asynchronously manage speaker groupings, giving users the power to listen to the football game on their living room speakers and rock out to their patio speakers at the same time. ***Being built for the current release.***

3.- Powerful web application that allows users to query music and instantly down-load that music. The project was done using nodejs and expressjs. ***A good idea to take into consideration.***

4.- This android application allows users to listen to a music playlist tailored to them based on their mood. Their mood is extracted using sentiment analysis on diary en-tries in the app. ***Being built for the current release.***

Notes and Future work. We are testing the phrases with other two members of the music application Startup, with the intention to perceive how much they differ in points of view as they have different background profiles. We are also working on identify more requirement patterns and test them in different Corpus of Readmes.

6 Conclusion

Mining existing information is becoming a strong ally in the process of requirements elicitation, since more and more information is being stored with open access in the web. GitHub as an open access repository for software projects is a strong candidate as an information source. However, to use GitHub information, which is scattered in thousands of projects, there is a need to compose a proper corpus, where mining heuristics could be applied.

This work describes the challenges and what has been done to build a Requirements Engineering oriented corpus taking GitHub project's Readme as a source.

With the results so far, we are more close to build a way of reusing unstructured, semi-structured and structured information linked to code, as to help the task of eliciting requirements-related information. As such, future work will focus on mining heuristics and validation of their application.

References

C. Arora, M. Sabetzadeh, Briand L. C., and F. Zimmer. 2014. Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. In *Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on*, pages 1–8.

Peter Pin-Shan Chen. 1976. The entity-relationship model&mdash;toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36.

Julio Cesar. S. do Prado Leite and A. Freeman. 1991. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, 17(12):1253–1269.

Fajar J. Ekaputra, Estefanía Serral, and Stefan Biffl. 2014. Building an empirical software engineering research knowledge base from heterogeneous data sources. In *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*, i-KNOW '14, pages 13:1–13:8. ACM.

Leah Goldin and Daniel M. Berry. 2015. Reuse of requirements reduced time to market at one industrial shop: a case study. *Requirements Engineering*, 20(1):23–44.

Bryan Goodrich, D Kurkiewicz, and Tyler Rinker. 2016. Bridging the gap between qualitative data and quantitative analysis.

Georgios Gousios. 2013. The ghtorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA. IEEE Press.

Julian Kupiec, Jan Pedersen, and Francine Chen. 1995. A trainable document summarizer. In *Proceedings* *of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 68–73, New York, NY, USA. ACM.

Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. 2002. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93.

Lynne M. Markus. 2001. Toward a theory of knowledge reuse: Types of knowledge reuse situations and factors in reuse success. *Journal of Management Information Systems*, 18(1):57–93.

Cade Metz. 2016. Triple play: Githubs code now lives in three places at once. *Wired*, Last accessed 08-14-2016.

Roxana L.Q. Portugal, Julio Cesar. S. do Prado Leite, and E. Almentero. 2015. Time-constrained requirements elicitation: reusing github content. In *Just-In-Time Requirements Engineering (JITRE), 2015 IEEE Workshop on*, pages 5–8. IEEE.

Antoinette Renouf. 2003. Webcorp: providing a renewable data source for corpus linguists. *Language and Computers*, 48(1):39–58.

M. Ridao, J. Doorn, and Julio Cesar. S. do Prado Leite. 2001. Domain independent regularities in scenarios. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 120–127.

J. Sinclair. 2005. *Corpus and Text - Basic Principles in Developing Linguistic Corpora: a Guide to Good Practice. Appendix: How to build a Corpus*. Oxford-Oxbow Books.