DECA: Development Emails Content Analyzer

Di Sorbo, Andrea ; Panichella, Sebastiano ; Visaggio, Corrado Aaron ; Penta, Massimiliano Di ; Canfora, Gerardo ; Gall, Harald

Abstract: Written development discussions occurring over different communication means (e.g. issue trackers, development mailing lists, or IRC chats) represent a precious source of information for developers, as well as for researchers interested to build recommender systems. Such discussions contain text having different purposes, e.g. discussing feature requests, bugs to fix etc. In this context, the manual classification or filtering of such discussions in according to their purpose would be a daunting and time-consuming task. In this demo we present DECA (Development Emails Content Analyzer), a tool which uses Natural Language Parsing to classify the content of development emails according to their purpose, identifying email fragments that can be used for specific maintenance tasks. We applied DECA on the discussions occurring on the development mailing lists related to Qt and Ubuntu projects. The results highlight a high precision (90%) and recall (70%) of DECA in classifying email content providing useful information to developers interested in accomplishing specific development tasks. Demo URL: https://youtu.be/FmwBuBaW6Sk Demo Web Page: http://www.ifi.uzh.ch/seal/people/panichella/tools/DECA.html

DOI: https://doi.org/10.1145/2889160.2889170

DECA: Development Emails Content Analyzer

Andrea Di Sorbo[1], Sebastiano Panichella[2], Corrado A. Visaggio[1],
Massimiliano Di Penta[1], Gerardo Canfora[1], Harald Gall[2]
[1]University of Sannio, Department of Engineering, Italy
[2]University of Zurich, Department of Informatics, Switzerland
disorbo@unisannio.it, panichella@ifi.uzh.ch, {visaggio,dipenta,canfora}@unisannio.it,
gall@ifi.uzh.ch

ABSTRACT

Written development discussions occurring over different communication means (e.g. issue trackers, development mailing lists, or IRC chats) represent a precious source of information for developers, as well as for researchers interested to build recommender systems. Such discussions contain text having different purposes, e.g. discussing feature requests, bugs to fix etc. In this context, the manual classification or filtering of such discussions in according to their purpose would be a daunting and time-consuming task.

In this demo we present DECA (Development Emails Content Analyzer), a tool which uses Natural Language Parsing to classify the content of development emails according to their purpose, identifying email fragments that can be used for specific maintenance tasks. We applied DECA on the discussions occurring on the development mailing lists related to Qt and Ubuntu projects. The results highlight a high precision (90%) and recall (70%) of DECA in classifying email content providing useful information to developers interested in accomplishing specific development tasks.

Demo URL: `https://youtu.be/FmwBuBaW6Sk`

Demo Web Page:

`http://www.ifi.uzh.ch/seal/people/panichella/tools/DECA.html`

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

Keywords

Unstructured Data Mining, Natural Language Processing, Recommender System

1. INTRODUCTION

When developers have to accomplish their working activities, they often analyze discussions appearing in development-specific communication means such as mailing lists, issue trackers and chats [10]. These means are useful for keeping track of development issues, possible problem solutions, or decisions taken [2, 11]. Generally speaking, they contain information and knowledge that can be decisive in making decision process.

The manual analysis of these discussions is time-consuming for several reasons: (i) a development email or a post on issue tracker contains a mix of structured, semi-structured, and unstructured information [1], (ii) the messages posted on such communication means may have different purposes (e.g., an issue report may relate to a feature request, a bug, or just to a project managment discussion, while such a classification can even be inaccurate [6]), (iii) sometimes emails or discussions are too long and the reader gets lost in unnecessary details, or (iv) pieces of information regarding the same topic are scattered among different sources (posts and emails) [10].

To help developers discarding unnecessary information, previous literature proposed approaches to classify emails' content (source code, text, stack traces, etc.) [1], as well as approaches aimed at generating summaries of emails [7, 13] and bug reports [15, 14]. However, none of the aforementioned approaches is able to classify paragraphs contained in developers' communication according to the *developers' intent*, and therefore filtering paragraphs useful for specific maintenance tasks, e.g., fixing bugs, adding new features, or improve existing ones.

This paper proposes a tool, named DECA (Development Email Content Analyzer), that uses Natural Language Parsing to automatically capture linguistic patterns and classify emails' content according to developers' intentions, such as asking/providing help, proposing a new feature, or reporting/discussing a bug. Specifically, the tool implements a mining approach defined in our previous work [4], which was originally conceived to mine useful information from mailing lists, but it can be also used to analyze development communication occurring in issue trackers and IRC chat logs. Moreover, DECA is available either in a Graphical User Interface (GUI) version, or as a Java library.

Our empirical evaluation indicated that DECA exhibits a high precision (90%) and recall (70%) overcoming performances of traditional approaches based on machine learning techniques [4].

Paper structure. Section 2 briefly describes the mining approach proposed in our previous paper [4] and its implementation. Section 3 shows how DECA works, while Section 4 provides some information about the performances of the tool. Finally, Section 5 concludes the paper.

2. OVERVIEW OF THE APPROACH

This section summarizes the approach behind DECA. Further details about the approach and its validation can be

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2889160.2889170

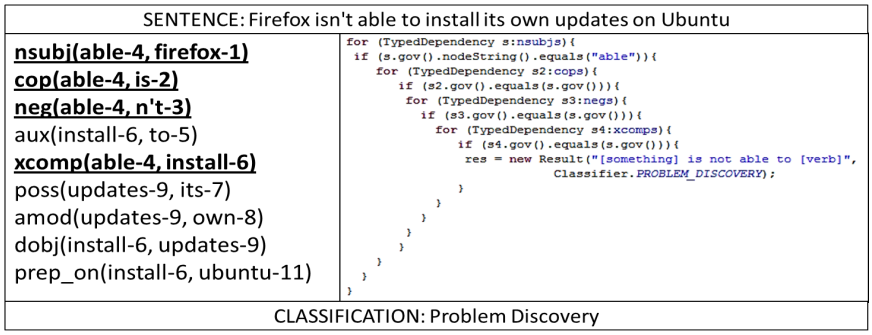

Figure 1: Implementation of a NLP heuristic

found in our previous research paper [4].

Table 1: Sentence Categories

Category	Description	Example Sentence
Feature Request	Linguistic patterns related to ideas/suggestions/ needs for improving or enhancing the product/service or its functionalities	*I think that adding the possibility to choose a keyboard layout during the automatic install process would be a good thing*
Opinion Asking	Linguistic patterns for requiring someone to explicitly express his/her point of view about something	*So what do you thing about that?*
Problem Discovery	Linguistic patterns related to issue definitions or unexpected behaviors	*Firefox isn't able to install its own updates on Ubuntu or on most any linux system*
Solution Proposal	Linguistic patterns used to describe possible solutions for known problems	*Wouldn't be a good idea to also include ikvm.net with classpath?*
Information Seeking	Linguistic patterns related to attempts to obtain information or help from other users	*Perhaps someone can tell me where the German sentences are stored*
Information Giving	Linguistic patterns exploited to inform/update other users about something	*Please note that we (Debian alsa mantainers) are working on getting OSS*

DECA classifies development emails content according to six categories describing the "intent" of the writer: *feature request, opinion asking, problem discovery, solution proposal, information seeking* and *information giving*. As described in [4], these categories were identified by a manual inspection of a sample of 100 emails, which have been firstly grouped according to the categories defined by Guzzi *et al.* in [5]. For each group of emails, significant sentences evoking, or suggesting the intent of the writer have been extracted and categories of sentences have been defined by using grounded theory.

For each category, Table 1 provides (i) a description and (ii) an example sentence. These categories are designed to capture the kind of information generally contained in messages regarding the development concerns.

When developers write about existing bugs, suggest features or solutions, ask for information or update other users about the project issues, within development email messages, they tend to use some recurrent linguistic patterns. These patterns present well defined predicate-argument structures related to *intentions*. DECA exploits these structures to automatically detect and categorize relevant text fragments for developers, within emails' content. Specifically, the tool analyzes messages at the **sentence-level granularity** because within a raw mail message some sentences could be relevant for software development purposes, while others could be not.

DECA has two main modules: the `Parser` and the `Classifier`. The `Parser` aims at preparing the text for the analysis. Firstly, it performs sentence splitting and tokenization, relying on the Stanford CoreNLP API [8]. Once the text is divided into sentences, the `Parser` creates, for each sentence, the Stanford Dependencies (SD) representation [3]. The Stanford Dependencies parser represents dependencies between individual words contained in sentences and labels each dependency with a specific grammatical relation (e.g., subject or direct/indirect object).

Such a representation is exploited by the `Classifier` to perform its analysis. We identified a set of 231 linguistic patterns [1] related to the sentence categories of Table 1. For each pattern, the `Classifier` implements a NLP heuristic to recognize it. Each NLP heuristic tries to detect the presence of a text structure that may be connected to one of the development categories, looking for the occurrences of specific keywords in precise grammatical roles and/or specific grammatical structures.

Figure 1 describes how the `Classifier` performs the classification for an example of sentence reported in the Ubuntu's development mailing list. The figure depicts the SD representation of the sentence (on the left-side) and the implementation of the NLP heuristic (on the right-side) able to detect the structure indicating (in the majority of cases) the disclosure of a problem/bug. The NLP heuristic has to analyze few typed dependencies (in the example of Figure 1, the code checks only the underlined dependencies) to detect the presence of a linguistic pattern. Once recognized a pattern, the `Classifier` returns the classification result to the `Parser`, which adds the result to a collection and provides the SD representation of the next sentence to the `Classifier`. The `Classifier` labels only the sentences that present known structures assuming that all other sentences are too generic or have negligible contents. At the end of this process, results are provided as output.

3. DECA IN ACTION

This section describes DECA's main features and provides examples of its usage.

The first version of DECA provides a practical GUI, which can be found in the zipped file `DECA_GUI.zip` available from the tool's webpage. The `README.txt` contained in the zipped file provides all the information to run the tool's GUI. To analyze discussions or messages the users can paste them in the text area of the GUI or alternatively, load them from a text file and press the `Recognize` button. When the recognition process is complete, DECA highlights all recognized sentences using different colors for different categories. Figure 2 shows the tool's GUI with an example of output. It is important to point out that DECA classifies exclusively the natural language fragments contained in the messages, since the `Classifier` can start its elaboration only when

[1] http://www.ifi.uzh.ch/seal/people/panichella/DECA_Implemented_Heuristics.pdf

the `Parser` is able to construct the SD representation for the sentence under analysis. Figure 3 depicts the tool's behavior for two examples of text fragments: the first one (on the top) contains a code snippet (for which the `Parser` is not able to construct the SD representation), while the second fragment (on the bottom) exhibits an error log containing some text in a natural language form (recognized by the `Parser` which can identify the dependencies structure).

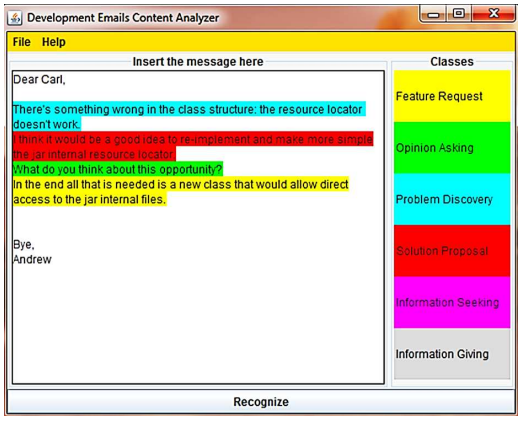

Figure 2: DECA's Interface

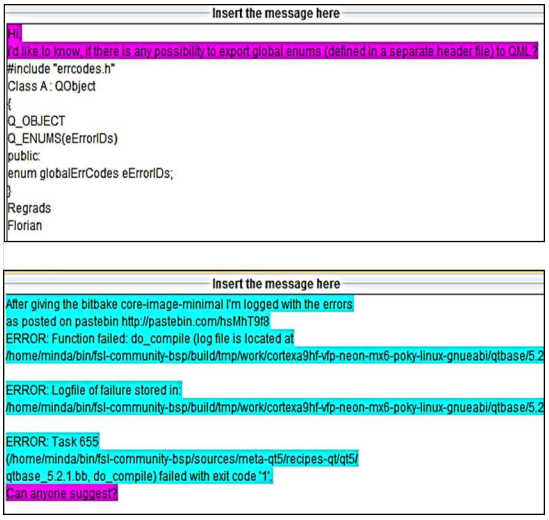

Figure 3: DECA's output in presence of code snippets and error logs

The second version of DECA is a Java API that provides an easy way to integrate our classifier in Java projects. Figure 4 shows an example of Java code that integrates the DECA's capabilities. To use it, it is necessary to download the `DECA_API.zip` from the tool's Web page, unzip it, and import the library `DECA_API.jar` as well as the Stanford CoreNLP libraries (which can be found in the lib folder contained in `DECA_API.zip`) in the build path of our Java project. Then, to use the DECA classifier, it is sufficent to import the classes `org.emailClassifier.Parser` and `org.emailClassifier.Result`, and instantiate the `Parser` through the method `getInstance`. The method `extract` of the class `Parser` represents the entry point to access to the tool's classification. This method accepts in input a String containing the text to classify and returns in output a collection of objects of the `Result` class. The `Result` class provides all the methods to access to DECA's classification results.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;

import org.emailClassifier.Parser;
import org.emailClassifier.Result;

public class Example{
  public static void main(String args[] throws Exception{
    Parser p = Parser.getInstance();
    BufferedReader br = new BufferedReader(
                        new InputStreamReader(System.in));
    System.out.println("Sentence: ");
    String text = br.readLine();
    ArrayList<Result> results = p.extract(text);
    if (!results.isEmpty()){
      for (Result r: results){
        System.out.println(r.getSentence()+" - "+
                           r.getSentenceClass());
      }
    }
  }
}
```

Figure 4: Using DECA as a Java library.

4. PERFORMANCE EVALUATION

Table 2: Results for Experiment I

CLASS	TP	FN	FP	PRECISION	RECALL	F-MEASURE
FEATURE REQUEST	30	43	0	1.000	0.411	0.583
OPINION ASKING	0	0	0	0.000	0.000	0.000
PROBLEM DISCOVERY	37	37	9	0.804	0.500	0.617
SOLUTION PROPOSAL	19	52	5	0.792	0.268	0.400
INFORMATION SEEKING	29	73	2	0.935	0.284	0.436
INFORMATION GIVING	13	43	3	0.813	0.232	0.361
TOTAL	128	248	19	0.871	0.340	0.489

Table 3: Results for Experiment II

CLASS	TP	FN	FP	PRECISION	RECALL	F-MEASURE
FEATURE REQUEST	32	26	7	0.821	0.552	0.660
OPINION ASKING	3	2	0	1.000	0.600	0.750
PROBLEM DISCOVERY	32	14	4	0.889	0.696	0.780
SOLUTION PROPOSAL	33	44	11	0.750	0.429	0.545
INFORMATION SEEKING	56	42	0	1.000	0.571	0.727
INFORMATION GIVING	41	20	4	0.911	0.672	0.774
TOTAL	197	148	26	0.883	0.571	0.694

Table 4: Results for Experiment III

CLASS	TP	FN	FP	PRECISION	RECALL	F-MEASURE
FEATURE REQUEST	46	8	9	0.836	0.852	0.844
OPINION ASKING	6	2	0	1.000	0.750	0.857
PROBLEM DISCOVERY	39	10	2	0.951	0.796	0.867
SOLUTION PROPOSAL	17	17	6	0.739	0.500	0.596
INFORMATION SEEKING	49	17	1	0.980	0.742	0.845
INFORMATION GIVING	26	24	2	0.929	0.520	0.667
TOTAL	183	78	20	0.901	0.701	0.789

We designed three experiments for a progressive assessment of the DECA's performances. In each experiment we increased the number of the implemented heuristics by using the false negatives obtained in the previous experiment. For more details see [4]. For assessing the tool's capabilities we rely on widely adopted metrics in the Information Retrieval field: precision, recall and F-measure. Tables 2, 3 and 4 show results achieved in the three experiments.

In the first experiment, we implemented 87 NLP heuristics to classify a test set of 100 emails randomly selected among messages exchanged by developers in May 2014, in the development mailing list of the Qt Project[2]. In the second experiment, we implemented 82 new NLP heuristics to classify a test set of 100 emails randomly selected among messages sent by developers in the development mailing list of the Qt Project during the year 2014. In the last experiment, we further implemented 62 new heuristics and used as

[2]http://qt-project.org

test set 100 emails randomly selected among messages sent from September 2004 to January 2005 from the development mailing list of the Ubuntu Linux distribution[3] achieving a global precision of 90% and a global recall of 70%. The experimental results show how the addition of new heuristics improves the effectiveness of DECA along the various experiments. Specifically, while the precision is always very high (it ranges between 87% and 90%) and stable for all the experiments, the recall increases with the addition of new heuristics from 34% to 70% (i.e., around two times) [4].

5. CONCLUSIONS

In this paper we presented DECA, a tool to automatically classify the content of development communication (e.g., emails) according to developers' likely intentions, such as requesting a feature, asking for an opinion, providing solutions, etc.. We built the tool on top of the Stanford English natural language parser [3, 8], applying a set of heuristics that have been defined on several training sets. DECA's performances have been assessed through an empirical study involving 300 messages from two different mailing lists: 200 emails from the QT project and 100 emails from the Ubuntu Linux distribution.

Results indicate that DECA achieves high levels of precision (90%) and recall (70%). Our experiments also demonstrate that implemented heuristics are able to classify intent of messages from mailing lists of different projects (both within-project and cross-project validations). As demonstrated in our previous work [4], DECA was also successfully used to improve the effectiveness of approaches aimed at mining source code documentation from developers communication [9, 16]. Moreover, DECA can be successfully used, in combination with Textual and Sentiment Analysis techniques, to retrieve feedback in user reviews of mobile app that are important for maintenance perspective (e.g. feature requests and all the requests to fix a bug) [12].

DECA could be used as a preprocessing support to discard irrelevant sentences within emails or bug reports summarization approaches. Furthermore, DECA could be used in combination with topic models for retrieving contents presenting the same *intentions* and treating the same topics in different kinds of development discussions, e.g., mailing lists, issue trackers and IRC chat logs. For example, such a combination could enable the possibility for a developer to retrieve all feature requests related to a given topic from different communication means in order to plan a set of change activities.

Acknowledgments

Sebastiano Panichella gratefully acknowledges the Swiss National Science foundation's support for the project *"Essentials"* (SNF Project No. 200020−153129).

6. REFERENCES

[1] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza. Content classification of development emails. In *Proceedings of 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 375–385, 2012.

[2] A. Begel and N. Nagappan. Global software development: Who does it? In *Proceedings of 3rd IEEE International Conference on Global Software Engineering, ICGSE 2008, Bangalore, India, 17-20 August, 2008*, pages 195–199, 2008.

[3] M.-C. de Marneffe and C. D. Manning. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser '08, pages 1–8, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[4] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 12–23, 2015.

[5] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen. Communication in open source software development mailing lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 277–286, Piscataway, NJ, USA, 2013. IEEE Press.

[6] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 392–401, 2013.

[7] D. Lam, S. L. Rohall, C. Schmandt, and M. K. Stern. Exploiting e-mail structure to improve summarization. In *ACM Conference on Computer Supported Cooperative Work (CSCW), New Orleans, LA*, 2002.

[8] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[9] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 63–72, June 2012.

[10] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol. How developers' collaborations identified from different sources tell us about code changes. In *Proceedings of 30th International Conference on Software Maintenance and Evolution (ICSME)*, pages 251–260. IEEE Computer Society, 2014.

[11] S. Panichella, G. Canfora, M. Di Penta, and R. Oliveto. How the evolution of emerging collaborations relates to code changes: an empirical study. In *Proceedings of 22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, pages 177–188, 2014.

[12] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can I improve my app? classifying user reviews for software maintenance and evolution. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 281–290, Sept 2015.

[13] O. Rambow, L. Shrestha, J. Chen, and C. Lauridsen. Summarizing email threads. In *Proceedings of HLT-NAACL 2004: Short Papers*, HLT-NAACL-Short '04, pages 105–108, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[14] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 505–514, 2010.

[15] S. Rastkar, G. C. Murphy, and G. Murray. Automatic summarization of bug reports. *IEEE Trans. Software Eng.*, 40(4):366–380, 2014.

[16] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora. Codes: Mining source code descriptions from developers discussions. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 106–109. ACM, 2014.

[3] http://www.ubuntu.com