# SOFTWARE DEVELOPMENT PRODUCTIVITY METRICS, MEASUREMENTS AND IMPLICATIONS

by

SHWETA GUPTA

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2018

THESIS APPROVAL PAGE

Student: Shweta Gupta

Title: Software Development Productivity Metrics, Measurements and Implications

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

| | |
|---|---|
| Boyana Norris | Chair |
| and | |
| Sara D. Hodges | Interim Vice Provost and Dean of the Graduate School |

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2018

© 2018 Shweta Gupta

THESIS ABSTRACT

Shweta Gupta

Master of Science

Department of Computer and Information Science

June 2018

Title: Software Development Productivity Metrics, Measurements and Implications

The rapidly increasing capabilities and complexity of numerical software present a growing challenge to software development productivity. While many open source projects enable the community to share experiences, learn and collaborate; estimating individual developer productivity becomes more difficult as projects expand. In this work, we analyze some HPC software Git repositories with issue trackers and compute productivity metrics that can be used to better understand and potentially improve development processes. Evaluating productivity in these communities presents additional challenges because bug reports and feature requests are often done by using mailing lists instead of issue tracking, resulting in difficult-to-analyze unstructured data. For such data, we investigate automatic tag generation by using natural language processing techniques. We aim to produce metrics that help quantify productivity improvement or degradation over the projects lifetimes. We also provide an objective measurement of productivity based on the effort estimation for the developer's work.

CURRICULUM VITAE

NAME OF AUTHOR: Shweta Gupta

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR
Mumbai University, Mumbai, India

DEGREES AWARDED:

Master of Science, Computer & Information Science, 2018, University of Oregon
Bachelor of Engineering, Computer Engineering, 2011, Mumbai University

AREAS OF SPECIAL INTEREST:

Software Development, Natural Language Processing, Data Mining, and Machine Learning

PROFESSIONAL EXPERIENCE:

Givens Associate, Argonne National Laboratory, 2017-2017

Associate Technical Lead, Software Developer, Zycus Infotech, 2013-2016

Software Developer, Infosys Limited, 2011-2013

ACKNOWLEDGMENTS

I would like to thank my advisor, Boyana Norris, without whom this work would not be possible. I would also like to thank my family and friends for their never ending support and inspiration.

For my parents and family.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

II. BACKGROUND DEFINITIONS AND TOOLS . . . . . . . . . . . . . . . 4

HTML Parsing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
Tokenizing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
Stemming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Stopword Removal . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Bag of Words (BoW) Model . . . . . . . . . . . . . . . . . . . . . . 7
K-means Clustering . . . . . . . . . . . . . . . . . . . . . . . . . 8
Latent Dirichlet Allocation (LDA) . . . . . . . . . . . . . . . . . 9
Term Frequency - Inverse Document Frequency (TF - IDF) . . . . 10
Word2Vec . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
Naive Bayes . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Semantic Parsing . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Anaphora Resolution . . . . . . . . . . . . . . . . . . . . . . . . 13
TextBlob . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
Natural Language Toolkit (NLTK) . . . . . . . . . . . . . . . . . . 15
Stanford NLP . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
Gensim . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

Chapter Page

SpaCy . . . . . . . . . . 16

Sci-kit Learn . . . . . . . . . . 16

Highcharts . . . . . . . . . . 16

Cytoscape . . . . . . . . . . 16

III. RELATED WORK . . . . . . . . . . 18

IV. METHODOLOGY . . . . . . . . . . 20

Data Collection . . . . . . . . . . 20

Preprocessing . . . . . . . . . . 23

Data Mining . . . . . . . . . . 25

V. RESULTS AND ANALYSIS . . . . . . . . . . 32

Data . . . . . . . . . . 32

Topic Labelling . . . . . . . . . . 32

Clustering . . . . . . . . . . 34

Visualizations . . . . . . . . . . 35

Sentiment Analysis . . . . . . . . . . 38

Profanity . . . . . . . . . . 41

Estimating Developer Effort . . . . . . . . . . 43

Developer - Code Graph . . . . . . . . . . 44

Developer - Contributor Graph . . . . . . . . . . 46

Miscellaneous . . . . . . . . . . 49

Chapter Page

VI. CONCLUSION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

REFERENCES CITED . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

LIST OF FIGURES

Figure Page

1. Google trends - VTK vs PETSc . . . . . 33
2. K-Means clustering. . . . . 34
3. Email and issue counts per developer. . . . . 36
4. Topics over time and count . . . . . 37
5. Developer sentiments over time. . . . . 39
6. Team sentiments over time. . . . . 40
7. Profane words. . . . . 41
8. Topics corresponding to profanity. . . . . 42
9. Developers' usage of profanity. . . . . 43
10. Individual developer effort estimate. . . . . 44
11. Developer sentiment vs effort. . . . . 44
12. Developers to code relations (scientific software). . . . . 45
13. Developer to code relations. . . . . 47
14. Developer to developer relation . . . . . 48

LIST OF TABLES

Table Page

1. Topic label analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
2. Graph measures for developer to code graph. . . . . . . . . . . . . . . . . 46
3. Graph metrics for PETSc developer to contributors graph. . . . . . . . . . 49

# CHAPTER I

# INTRODUCTION

Software development is a high-expertise, high-effort activity that occurs in industry, science, engineering, and many other areas. Software engineering researchers have long attempted to quantify the human effort expended to produce artifacts. Because of the inherent difficulties in measuring all the different things that would comprise such "effort", existing approaches are necessarily based on incomplete and noisy data.

Scientific software development traditionally focuses on functionality, accuracy, and performance, and to a much lesser extent on the productivity of developers. Some new efforts such as the DOE IDEAS (Productivity [2011]) project aim to achieve a better understanding of HPC developer productivity and explore methods for improvement through existing and new software engineering practices. An important first step in this quest is to develop metrics that help understand the factors that affect developer productivity.

We intend to analyze HPC software Git repositories, issue tracking data, and developer/user mailing lists to estimate productivity metrics that can be used to better understand and potentially improve development processes. Issue tracking systems have not yet gained wide adoption in scientific computing, although they are commonly used in software projects elsewhere. When used, issue trackers enable developers and users to submit and track different issues, such as bugs, feature requests, enhancements, or any other tag the developers wish to define for their project. However, many scientific software projects began long before issue tracking became available, so a large part of their development history is saved in mailing list archives. For such data, we investigate automatic tag generation by using natural language processing techniques. We also evaluate some non scientific software projects including Linux (Torvalds [1999]), Couch Potato (Burger [2010]), Famous (Valdman [2014]), l2met (Smith [2012]), KindleToPdf (Hou and Aivazian [2011]) etc to understand the difference between the traditional projects and scientific projects.

In this work we are considering the following scientific software libraries, which are funded in part through government research grants, but also depend on unpaid contributors.

– Portable, Extensible Toolkit for Scientific Computation (PETSc): a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/Tao) also contains the Tao optimization software library (Balay et al. [2018]).

– Supercomputer PACKage manager (SPACK): a flexible, configurable, Python-based HPC package manager, used for automating the installation and fine-tuning of simulations and libraries. It operates on a wide variety of HPC platforms and enables users to build many code configurations (Gamblin et al. [2015]).

– FLASH: a multiphysics multiscale simulation code with a wide international user base. It is the product from The Flash Center for Computational Science at the University of Chicago which has been home to several cross-disciplinary computational research projects in its 20-year existence (at University of Chicago [2011]).

- The Visualization Toolkit (VTK): an open-source, freely available software system for 3D computer graphics, image processing, and visualization. It consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods, as well as advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation (Schroeder et al. [2006]).

# CHAPTER II

# BACKGROUND DEFINITIONS AND TOOLS

Scientific software is developed to answer scientific research questions. Most of the projects are started as a proof of a concept until they actually evolve. They are developed by domain experts, such as physicists and applied mathematicians, who are highly skilled in their fields but are essentially not computer science experts. Scientific software projects typically do not employ traditional software development paradigms such as formal design, testing, versioning or documenting.

Moreover, as the complexity of software grows, it becomes equally important to pay as much attention to software reusability, efficiency, robustness, and user-friendliness as to developing new functionality. The growing research software complexity require more research towards understanding ways of improving both software quality and productivity.

Scientific software communities are different from non-research software development communities. A lot of time the focus is on library or application development in a super fast agile manner rather than having to manage or maintain the software products. There is always a lot of discussion with stakeholders on enhancements and contributions from the community. Most of the times the collaborators are geographically distributed. As a result of a lot of discussions written in different formats. Moreover, as the tools and technology evolve, there is little or no refactoring or restructuring. Thus, mining of developer discussions becomes essential. Machine learning, data mining tools, and models can help us perform unsupervised classification and clustering. The preliminary steps to perform the analysis is data cleaning and parsing. There are a lot of open source libraries available to accomplish data analysis. We have listed some of these techniques and libraries below.

## HTML Parsing

Data collection is one of the major steps in data mining. HTML parsing is used for web scraping. By doing this the data is downloaded, extracted and presented in a format that a developer can easily make sense of it.

HTML is a syntax specific language. Every tag serves as a block in a web page. If the websites adhere to the HTML syntax it should be easy to scrape. Using python it's easy to do web scraping using BeautifulSoup library Nair [2014]. A regular HTTP request is made to the server which returns HTML page as the response. The response is then parsed to BeautifulSoup format so we can use BeautifulSoup to work on it. Using functions on the BeautifulSoup object like "find_all" and "get" can help in getting the desired object.

## Tokenizing

Text tokenizing is a way of splitting the text to tokens. A token can be either be a word, a sentence or a paragraph. In this work, we have used Python NLTK tokenizer Loper and Bird [2002] and StanfordNLP Manning et al. [2014] Tokenizer. Both these instances use Penn Treebank 3 (PTB) tokenization Marcus et al. [1999] which works on simple rules for punctuations, double quote, parsed corpus etc. The rules are mentioned here Taylor et al. [2003].

Example:

I/P :: "Since this is a big code, it has tons of output."

O/P :: [Since, this, is, a, big, code, it, has, tons, of, output,.]

## Stemming

Stemming is a process which essentially removes the morphological affixes from words so that we are left with the root word. Both NLTK and StanfordNLP use Porter stemmer algorithm Porter [1997] to remove the common morphological and inflexional endings from words in English. It is a rule-based stemmer that is applied to the given word in a particular sequence.

Example:

I/P :: "Since this is a big code, it has tons of output."

O/P :: "Sinc thi is a big code , it ha ton of output."

## Stopword Removal

Usually, most frequent words don't make any sense when we do information retrieval. In such a case it's necessary to remove such words. The process of removal of most frequent words from the given text is called stopword removal. Every NLP library has a set of predefined stop words like "is", "of", "the", etc. There is no universal list of stop words in NLP research, however, the NLTK module contains a list of stop words. These words can be hardcoded or the library may have defined corpus for the same. Eventually, each word in the given text is matched with the dictionary to determine whether it is to be kept or removed.

Example:

I/P :: "Since this is a big code, it has tons of output."

O/P :: [Since, big, code, tons, output]

## Bag of Words (BoW) Model

In order to simplify representation, we use Bag of words model. It is one of the most common representations used for the various machine learning algorithm as input. The problem when we model an algorithm is that the input is messy. Most of the time they prefer a fixed length input and output. The text cannot be feed directly as raw. They need to be converted to a vector representation. Essentially, the vector to the actual words is also necessary as to make sense of the results.

Bag of words (BoW) is very simple and flexible mechanism to do so. A BoW contains two things - A vocabulary of the given word and measure of the presence of word (Brownlee [2017]). It is called "bag" as the information about the order or structure of words is not maintained. Consider an example: A text snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.

it was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness

The scoring would be like ("it" : 1), ("was" : 1), ("the" : 1), ("best" : 1), ("of" : 1), ("times" : 1), ("worst" : 0), ("age" : 0), ("wisdom" : 0), ("foolishness" : 0)

The documents would look as follows:

"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]

"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0] “it was the age of foolishness” = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

## K-means Clustering

K-means is an unsupervised machine learning algorithm which helps in finding a fixed set of data clusters. A cluster is essentially the group of data points which have same features. When using K-means we start with real or imaginary centers for each cluster. Then every point in the dataset is added to a cluster based on the standard defined distance measure. This process is then repeated until a fixed number of times or until the centers converge.

Example:

I/P datapoints: A1(3,10), A2(4,6), A3(9,5), B1(2,8), B2(8,5), B3(6,6), C1(3,3), C2(5,7), C3(6,8)

Lets say we have 3 centers (A1,B1, C1) and we take square error distance as our measure.

After first iteration we have clusters as

Cluster 1: [(6,8), (3,10)]

Cluster 2: [(5,7), (2,8), (4,6)]

Cluster 3: [(3,3), (6,6), (8,5), (9,5)]

And the new centres for cluster are: [(4.5, 9), (3.67, 7), (6.5, 4.75)]

Following are the basic distance measures:

*Euclidean Distance*

The Euclidean distance between two points is the length of the path connecting them

$$\sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

*Manhattan Distance*

The Manhattan Distance between two given points is the absolute distance of their Cartesian points

$$|x_1 - x_2| + |y_1 - y_2|$$

*Cosine Similarity*

Given two non - zero vectors cosine similarity is the measure of cosine angle between them

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i}\sqrt{\sum_{i=1}^{n} B_i}}$$

**Latent Dirichlet Allocation (LDA)**

Latent Dirichlet Allocation is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar (Wikipedia contributors [2018a]). It is the most widely used topic modelling algorithm. A document can be expressed as a mixture of topics. Each document consists of words that contribute to a given topic and it can be predicted by generating the probability distribution. Given the dataset and LDA backtracks it tries to figure out the most likely topic for that document. An open source library to perform LDA in python is Gensim Řehůřek and Sojka [2010]. It scales well on a large text corpus. Example Chen [2011]:

Suppose you have the following set of sentences:

I like to eat broccoli and bananas.

I ate a banana and spinach smoothie for breakfast.

Chinchillas and kittens are cute.

My sister adopted a kitten yesterday.

Look at this cute hamster munching on a piece of broccoli.

LDA is a way of automatically discovering topics that these sentences contain. For example, given these sentences and asked for 2 topics, LDA might produce something like

Sentences 1 and 2: 100% Topic A

Sentences 3 and 4: 100% Topic B

Sentence 5: 60% Topic A, 40% Topic B

Topic A: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, (at which point, you could interpret topic A to be about food)

Topic B: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, (at which point, you could interpret topic B to be about cute animals)

### Term Frequency - Inverse Document Frequency (TF - IDF)

TF-IDF is a numerical statistic which reflects how important a word is to a document in the given dataset (Wikipedia contributors [2018b]). The importance of a word increases proportionally to the number of times a word appears in a document. As the name suggests it composes of two terms, Term Frequency (TF) and Inverse Document Frequency(IDF). TF is defined as the frequency of a given word in a document. In order to normalize the occurrence of any word with respect to the length of a document it is divided by the total number of terms in the document. Thus, TF(x) = (Number of times “x” appears in a document) / (Total number of terms in the document).

IDF on other hand computes the importance of a term. In order to scale down the importance of frequently occurring stop words, we take the logarithmic value of the term.

Thus, IDF(x) = log(Total number of documents / Number of documents with term “x” in it).

Example:

Consider a document has 100 words and the word “dog” appears 8 times. The TF for “dog” is (8/100) = 0.08. Now, let’s assume we have 1000 documents and “dog” appears in 100 of them. Thus IDF = log (1000/100) = log(10) = 1. So, the tf-idf is given as 0.08 * 1 = 0.08.

## Word2Vec

A word2vec model is a word embedding model. It takes the text as input and returns a vector space of the specified dimension. The word vector represents a word in the space in such a way that the words that share common contexts in the corpus are close to each other. Word2Vec is a shallow neural network with three layers; the input, output and a hidden layer. An input to a neural network is a sequence of numbers that represents the given text. This representation is called “One Hot Encoding”. The word2vec model can be based on either of two models Continuous Bag of Words and Skip-gram model.

### *Continuous Bag of Words (CBOW)*

CBOW model predicts the base word, given the surrounding words. We define the window size while feeding data to the neural network. Consider a sentence as “cat climbs tree for the first time ever” and the window size is 3, the input and target will be [(cat, tree), climb] where climb is the target word.

### *Skip-gram*

Skip-gram model predicts the context given a word. There is just one input feed to the model which is one hot encoded. In response, we get a list of words which are more likely to share context with it. Consider if we give an input as “Climbed” and “Tree” and “Cat” are in context, then the output will be cat and tree.

If the dataset size is smaller then, Skip gram performs better than the CBOW. CBOW needs a lot more training based on the text and window size.

## Naive Bayes

Naive Bayes is a classification algorithm which is based on the Bayesian Theorem. It assumes that a presence of a feature is unrelated to any other feature. Thus, it finds the probability of an event occurring given that the other event has already occurred.

$$P(C_k/B) = \frac{P(B/C_k)P(C_k)}{P(B)}$$

where K is the number of possible outcomes for feature C and B is essentially a vector of features.

## Semantic Parsing

Semantic parsing is a task of converting the natural language task to a logical form. A logic form could be a machine-understandable representation of a database query. Most of the application of semantic parsing is in code generation and question answering applications.

### *Lambda Dependency-Based Compositional Semantics (Lambda DCS)*

Lambda DCS works on the fact that it has a knowledge base of asserts in the background. Each entry in the knowledge base is of type Entity, Property, Entity. This can be visualized as a directed graph where nodes are entities and edges represent assertions in the knowledge graph. Each rule is written as a lambda operator and can be evaluated as an unary case, binary case, join, intersection, union, negation or higher-order functions Liang [2013].

Example: Consider an entity like Seattle, it can be represented in the unary form as

$$\lambda x.[x = Seattle]$$

### *Combinatory Categorial Grammar (CCG)*

In CCG the verbs are associated with a syntactic "category" which identifies them as functions and specifies the type and directionality of their arguments and the type of their result.

## Anaphora Resolution

The problem resolving pronoun is called as Anaphora Resolution. For humans, it's an easy task to resolve a pronoun with the noun knowing the text preceding it. For machines, it's one of the most challenging tasks. It is considered as one of the major Black Hole in the NLP domain. There are two different libraries that address this issue

#### *Stanford Deterministic Coreference Resolution System*

This system implements the multi-pass sieve coreference resolution (or anaphora resolution) system written in Java.

#### *Neural coref*

As the name suggests, Neural coref is a neural network model based on SpaCy parser inspired by Kevin Clark's and Christopher D. Mannings Deep Reinforcement Learning for Mention-Ranking Coreference Models(Clark and Manning [2016]). It extracts the potential mentions, that is, words that are referring to real entities. Based on the predefined rules and parsing the dependency tree, it trains the model. Further, the most likely feature is selected from the input text and a pairwise ranking is given.

### TextBlob

Textblob is a simple Python library for processing textual data. It provides a simple API for diving into common Natural Language Processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. It's a simple, easy to understand, well-documented library with many features for performing NLP tasks. We used this library to perform sentiment analysis. The TextBlob sentiment analysis we used was powered by Naive Bayes Classifier. The classifier is trained on Movie Review dataset available in NLTK for sentiment analysis. The accuracy of this model is 72%. Although it's not very high, it is still very useful.

### Natural Language Toolkit (NLTK)

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum (Loper and Bird [2002]).

It comprises of suites of program modules, data sets, tutorials and exercises, covering symbolic and statistical natural language processing. NLTK is written in Python and distributed under the GPL open source license.

### Stanford NLP

The Stanford NLP Group makes Natural Language Processing tools, modules, datasets which provide statistical NLP, deep learning NLP, and rule-based NLP tools for major computational linguistics problems. The packages are widely available to be used in industry, academia, and government (Manning et al. [2014]).

### Gensim

Gensim is the most robust, efficient and hassle-free piece of software to realize unsupervised semantic modelling from plain text.It uses NumPy, SciPy and optionally Cython for performance. Gensim is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms, which differentiates it from most other scientific software packages that only target batch and in-memory processing.(Řehůřek and Sojka [2010])

### SpaCy

SpaCy is yet another open source software written in Python and Cython for Natural Language Processing. It has features like neural models for tagging, parsing and entity recognition. It scales well on large data sets and is highly memory managed (Honnibal and Montani [2017])

### Sci-kit Learn

Scikit is a simple and efficient tool for data mining and data analysis. It is accessible to everybody and reusable in various contexts. It is built on NumPy, SciPy, and matplotlib. It contains many states of the art algorithms that are easy to use with high-level APIs and well-maintained documentation (Pedregosa et al. [2011])

### Highcharts

Highcharts are highly interactive JavaScript charts used for the web pages. Highcharts supports line, spline, area, areaspline, column, bar, pie, scatter, gauge, arearange, areasplinerange and column range chart types. It is easy to use with high-level APIs and well-maintained documentation.

### Cytoscape

Cytoscape (P et al. [2003]) is an open source software project for integrating biomolecular interaction networks with high-throughput expression data and other molecular states into a unified conceptual framework. It is most powerful when used in conjunction with large databases of protein-protein, protein-DNA, and genetic interactions that are increasingly available for humans and model organisms. In this work we are using Cytoscape to analyse large networks such as inter-developer interactions and developer commits directed graph.

# CHAPTER III

# RELATED WORK

Software productivity is a measure which can help control and improve software development processes. It has been subject to active research for a long time. Martin Fowler (Fowler [2003]) mentions in his blog that the software industry lacks the ability to measure some of the basic elements of the effectiveness of software development because productivity is something you determine by looking at the input of an activity and its output. So to measure software productivity you have to measure the output of software development - the reason we can't measure productivity is because we can't measure output. However, there are various communities and groups trying to measure and determine what factors affect software development.

It is very interesting to explore the largest open source software repository - GitHub. A lot of factors can potentially be including in estimating the productivity developers and teams in open source software projects. Tsay et al. [2014] investigate how contributions are affected by discussions on GitHub. On the other hand, Ray et al. [2014] study the programming language as a factor for code quality. They use Negative Binomial Regression to model the non-negative counts of project attributes such as the number of commits.

Some researchers study the psychological aspects of development based on these repositories. For example, Thung et al. [2013] create the network structure that reflects developers' influence on GitHub. They identify the most influential developers and projects using the PageRank algorithm. Understanding how developers and projects are actually related to each other on a social coding site is the first step towards building tools to aid social programmers in performing their tasks more efficiently. Guzman et al. [2014] study Git users' sentiments in the commit comments to analyze emotions and related factors which could be a factor in managing programming teams.

Vasilescu et al. [2015a] try to understand the perception of diversity on GitHub. Contributors can be professional developers or volunteers of varied personalities, educational and cultural backgrounds, age, gender, and expertise. Diversity on GitHub can be used as complementary data to quantitatively study of developer's behavior which can affect and impact software productivity. Another research paper by Vasilescu et al. [2015b] studies gender and tenure diversity, aiming for better outcomes in recruiting and performance.

The book Social Informatics by Jarczyk et al. [2014] addresses project quality and a project's influence on other projects by defining some metrics. Their first metric is based on the number of stars, the other is on survival analysis techniques applied to issues reported on the project by its users. The book on Software Quality Engineering by Kan [2002] describes various metrics associated with the software life-cycle: in-process, end-product, and maintenance.

# CHAPTER IV

# METHODOLOGY

Scientific projects evolve very quickly and can be active for decades. There are various time-dependent factors that matter in the project's evolution and popularity. Traditional software productivity analysis cannot be applied easily to these projects. Most established metrics are static and rely purely on code factors. We are also trying to discover other potential factors that affect developer and team productivity, such as the changing budget of the project, the effects of the research funding environment, etc.

In this work, we are trying to understand how the successful active projects differ from the traditional archived project. Can we predict the lifespan of a project? How does a developer sentiment affect the team? Does the time variant sentiment also affect its productivity? Does the productivity of the team as a whole depend on one or a few core developers? How impactful is each core developer? Unlike in product-focused software projects, since there are no easy replacements for the same work, what happens if the core developer stops contributing?

Following are the steps that are necessary to perform data analysis that aims to help answer these and future questions.

## Data Collection

### *Emails*

Most of the open source software libraries have publicly available mailing lists hosted by Mailman (Warsaw et al. [1999]) servers. In order to download the data, we needed to perform web scraping. We used Pythons BeautifulSoup to do so. We considered the following are the projects and their repositories:

- PETSc
    * petsc-maint(https://lists.mcs.anl.gov/mailman/private/petsc-maint/)
    * petsc-users (http://lists.mcs.anl.gov/pipermail/petsc-users/)
    * petsc-dev (http://lists.mcs.anl.gov/pipermail/petsc-dev/)
- Linux
    * Linux-Kernel Archive(http://lkml.iu.edu/hypermail/linux/kernel)

## *Github Commits*

Git (Chacon and Straub [2014]) is a free and open source version control system. It can handle all size of projects with speed and efficiency. Git commit records all changes to the repository. Ones with multiple commits to the repository the log of commits can be done. The following git commands provide data on all commits:

```
git clone ⟨repository_url⟩
git log –abbrev-commit –decorate –all –numstat -p
```

These commands gives author, file name, commit diffs and commit date. Since it is the fastest way to pull commit history, we performed this for most of the repositories mentioned below

- PETSc
- LINUX

– COUCHPOTATO

– FAMOUS

– KINDLETOPDF

– VTK

– l2met

### *GitHub Issues*

Git issues are a great way to keep track of tasks, enhancements, and bugs. They are similar to emails but are shared with the entire team instead of specific contributor/developer. Github's issue tracking tracks collaboration, references, and excellent text formatting, making it ideal for your research.

GitHub issues can be downloaded as JSON objects using the well-documented GitHub REST API v3 Developer [2017] to Github's web services. The web services are secured and hence one needs to have a GitHub account for access. We cloned the following repositories' issue trackers.

– SPACK

### *Bitbucket*

Bitbucket is a web-based version control system based on Git. Some projects like PETSc have their version control managed using Bitbucket. The commit messages and code diffs were downloaded in a similar fashion to Github's data.

#### *SVN*

Apache Subversion (SVN) is another version control system under the Apache Licence. With SVN users can create self-hosted private repositories. FLASH uses SVN. In order to download commit data and messages, we needed to have access to their private repository. Once the access was granted, we created a Git local repository so that we could use the same parser as for other projects. We used git-scm (Chacon and Long [2005]) for the conversion. Finally, we were able to parse the data in the same format.

### Preprocessing

In this section we overview the data preprocessing steps we implemented.

#### *Selection*

Most of the emails are threads, i.e. they include one or more replies to the original email. We must distinguish the content of the new email from the old email. The relevant content new to the current email was extracting by using regular expressions and the fact that most of the previous content was started with prefix "⟩" and was indented.

Most of the time emails start salutations and end with the author's signature. We created a dictionary of the most common salutations and authors and filtered it out from the text. In addition, we parsed the text to find most frequent developers and used them in the list of stopwords so that the models are not biased towards them.

Preprocessing steps like tokenizing, removing stopwords and removing non-ASCII characters were performed on unstructured text. Since stemming changes the meaning of the word and we wanted to determine what part of speech it is, we choose not to do it.

As an illustration of what our inputs look like, below are a couple of examples of the structure of a FLASH email object and a PETSc commit object.

```
{
    "_id" :"597d78cd785d391f2896270b",
    "author" : "Jason Galyardt",
    "text" : "Dear FLASH developers, My apologies -- I reported the same bug that Mark found. There
     ↪   are two new bugs in source/physics/Hydro/HydroMain/unsplit/Hydro_computeDt.F90 on lines
     ↪   188 and 190, also having to do with array indices: !~~~~ jmS = max(blkLimits(LOW,IAXIS),
     ↪   jmS) !... jpS = min(blkLimits(HIGH,IAXIS), jpS) !~~~~ The 'IAXIS' on both lines should be
     ↪   'JAXIS', I believe. Here are the corrected lines: !~~~~ jmS = max(blkLimits(LOW,JAXIS),
     ↪   jmS) !... jpS = min(blkLimits(HIGH,JAXIS), jpS) !~~~~ Sorry about the confusion. Again,
     ↪   these bugs are also in source/physics/Hydro/HydroMain/unsplit_rad/Hydro_computeDt.F90.
     ↪   Best Regards, Jason  On Fri, Feb 12, 2016 at 3:52 PM, Jason Galyardt <jason.galyardt at
     ↪   gmail.com> wrote: > Dear FLASH developers, > > In looking at
     ↪   source/physics/Hydro/HydroMain/unsplit/Hydro_computeDt.F90 > to correct a bug reported to
     ↪   the flash-users list by Mark Richardson, I > noticed another bug. Line 239 in the FLASH
     ↪   4.3 version of this file reads > as follows: > > if (NDIM > 2) dt_ltemp = >
     ↪   max(dt_ltemp,(abs(U(VELZ_VAR,i,j,k)-uzgrid(j))+sqrt(cfz2))*delzinv) > > However, I believe
     ↪   the index of the uzgrid array is incorrect; the index > 'j' should be 'k', yielding: > >
     ↪   if (NDIM > 2) dt_ltemp = >
     ↪   max(dt_ltemp,(abs(U(VELZ_VAR,i,j,k)-uzgrid(k))+sqrt(cfz2))*delzinv) > > I've not noticed
     ↪   any problems traceable to this bug, though I can verify > that the corrected version is
     ↪   functional. Given that Mark Richardson's bug > was first identified in
     ↪   unsplit_rad/Hydro_computeDt.F90, the bug identified > above likely affects the unsplit_rad
     ↪   version of this routine as well. > > Best Regards, > > Jason > > ---- > Jason Galyardt >
     ↪   Department of Physics and Astronomy > University of Georgia > > ",
    "date" : "2016-02-12T15:25:20.000Z",
    "pid" :"597d78c8785d391f2896270a",
    "title" : "Bug in Unsplit Hydro_computeDt",
    "url" : "http://flash.uchicago.edu/pipermail/flash-bugs/2016-February/000392.html",
    "content" : "Dear FLASH developers,  My apologies -- I reported the same bug that Mark found.
     ↪   There are two new bugs in source/physics/Hydro/HydroMain/unsplit/Hydro_computeDt.F90 on
     ↪   lines 188 and 190, also having to do with array indices:  !~~~~ jmS =
     ↪   max(blkLimits(LOW,IAXIS), jmS) !... jpS = min(blkLimits(HIGH,IAXIS), jpS) !~~~~  The
     ↪   'IAXIS' on both lines should be 'JAXIS', I believe. Here are the corrected lines:  !~~~~
     ↪   jmS = max(blkLimits(LOW,JAXIS), jmS) !... jpS = min(blkLimits(HIGH,JAXIS), jpS) !~~~~
     ↪   Sorry about the confusion. Again, these bugs are also in
     ↪   source/physics/Hydro/HydroMain/unsplit_rad/Hydro_computeDt.F90.  Best Regards,  Jason   On
     ↪   Fri, Feb 12, 2016 at 3:52 PM, Jason Galyardt <jason.galyardt at gmail.com>",
    "prediction" : "bug"
}
```

```
{
    "_id" : "d3a0d25",
    "author" : "Alp Dener",
```

```
    "commits" : [
        {
            "deletions" : "1",
            "diffs" : [
                {
                    "spacy" : 1.00000003488181,
                    "tag" : "change",
                    "text2" : "{\\bf Todd Munson \\\\ Jason Sarich \\\\ Stefan Wild \\\\ Steven
                     ↪   Benson \\\\ Lois Curfman McInnes}",
                    "max_char" : 84,
                    "text1" : "{\\bf Alp Dener \\\\ Todd Munson \\\\ Jason Sarich \\\\ Stefan Wild
                     ↪   \\\\ Steven Benson \\\\ Lois Curfman McInnes}",
                    "diff" : 0.925820099772552,
                    "_id" : "d3a0d25_3",
                    "diff_eff" : 6.23111161910567
                }
            ],
            "filename" : "src/docs/tao_tex/manual/cover.tex",
            "diff_eff_avg" : 0.0110216678544351,
            "additions" : "1",
            "diff_eff" : 6.23111161910567
        },
        {
            "deletions" : "1",
            "additions" : "1",
            "filename" : "src/tao/unconstrained/impls/lmvm/lmvm.c"
        }
    ],
    "diff_eff_avg" : 0.925820099772552,
    "date" : "2018-03-05T19:44:07.000Z",
    "diff_eff" : 6.23111161910567
}
```

## Data Mining

In this section we briefly overview the different data mining methods we applied.

### *Topic Labelling*

In order to identify whether the email is primarily about a bug or a feature request, we performed topic labeling. We used Latent Dirichlet allocation in python Gensim to perform topic labeling.

Each email was considered as the document to analyze. After preprocessing the documents, they were ready to be fed to LDA model. However, LDA requires some sort of text encoding, so that the input is a vector of numbers. In order to convert words to vector, we used Bag of word representation. Gensim creates a dictionary of encapsulates the mapping between normalized words and their integer IDs.

Because the vocabulary size is huge we restricted the dictionary to contain 200000 words only. Once the bag of words representation is ready, it is fed as input to LDA model. The LDA model gave the probability distribution of words which contributed to the topic. Since the topics are unlabelled we get an output as *topic1*, *topic2* and corresponding word association along with the probability of how much the word contributes towards the topic. We then used human judgment to conclude that the words such as "error ", "lead ", and "experiment " represent **Bugs** and words such as "would ", "like ", "change " represent **Features**.

Once we label the topics with **Bug** and **Feature** labels, we need to tag each email. So, each document was then preprocessed and converted to Bag-of-word representation. For Bag-of-word each document is converted into a vector representation. The number of items in the vector representing a document corresponds to the number of words in the vocabulary. The larger the vocabulary, the longer the vector representation, hence the preference for smaller vocabularies in the previous section. The next step is to feed again to LDA model previously generated. This process gives out the probability distribution of the document to the topics. The higher the probability, the more likely it is that the document is intended to be related to that topic. Thus, we were able to label all documents as **Bug** or **Feature**.

### *Clustering*

Email chains are usually supposed to have the same behavior in terms of thr number of emails exchanged in the mail chains or the topic of discussion is relevant to the team. This is called its normal behavior. However, there might be some deviation from the standard normal pattern. This is called an anomaly. To detect the unusual pattern we decided to perform unsupervised clustering. In particular, we used TF-IDF over the titles to perform k-means clustering where the number of clusters is two.

### *Data Visualization*

The implicit features are frequently overlooked while performing complex analysis. Simple patterns can depict very useful and meaningful information. Thus, after preprocessing the data, we decided to plot emails exchanges per contributor over the years, email density over time per unique chain and author to bug/feature ratios.

### *Sentiment Analysis*

In order to track the overall mood of developers with respect to a project average, we projected the sentiment analysis on emails and issues discussed. We used the TextBlob sentiment analysis feature to perform sentiment analysis. The interesting patterns we wanted to evaluate was, how does the open source non-funded project work against scientific projects which are government funded. How does the course of project change when the funding changes? Does each developer's sentiment matter equally? Can we detect/predict when a the project encounters a difficult time? Does male to female ratio matter when it comes to sentiment analysis? Does a lot of work pressure make people negative? How productive is a more negative community compared to a more positive one?

In this work, we choose to explore only some aspects and some research software and non-research open-source projects, such as SPACK, PETSc, FLASH, VTK, and LINUX. Standard preprocessing steps are performed on the content of the unstructured text and then using the analyzer the probability of the text being positive or negative is calculated. Then, based on the higher percentage the document is labeled as positive or negative.

### *Profanity*

Psychology researchers believe that a human swears for reasons such as to show power and control, express humor, increase peer bonding, etc. Some studies claim that swearing is the sign of intelligence (Stephens [2017]). In order to understand how profanity affects the current projects and some old projects, we did profanity analysis in Python. Since there is not a predefined list of profane words and no standardized dictionary, we decided to create our own profane dictionary of words.

The list contained a total of 438 words which excluded word relevant to the domain. For example, the daemon was excluded from the dictionary as it is an important keyword in computer science domain.

The profanity analysis is performed by preprocessing the text. Once that is done, using a word tokenizer we check if each word is present in the profanity dictionary. If the word exists we mark that email to contain profane word and created a list of all profane words in the text. After generating the profanity list we create a dictionary model which essentially contains the profane words and the word count associated with it. We then rank the dictionary for word count from higher to lower. Based on a certain threshold for each project we then make the judgment for the topic with which profanity is most associated with. Currently, we have explored profanity analysis on big repositories like LINUX and even on the smaller ones like SPACK, PETSc, and FLASH.

### *Developer Effort Estimate*

There are various metrics that defines effort estimation. For example addition of total lines of code or deletion of it, number of characters added/deleted, etc.

We decided to estimate effort based on fine-grained change analysis instead of the more typical file-based changed lines metrics. Each commit may have one or more changed files, and each file may have one or more sets of changed lines, which we consider individually. After cloning the repository we generate the log file using git. A parser processes the log to generate a set of fine grained commit changes. Each addition/deletion is 100% effort whereas an update is evaluated based on the significance of the change computed through the similarity analysis discussed below.

- Similarity using a vector method

  For each of the text to compare, a number vector is calculated based on the frequency of words. Then for both the vectors say A and B, the difference is calculated as the cosine similarity. In this process, pure Python code is used to calculate similarity.

- Similarity using cosine difference on TF-IDF

  Sci-kit Learns TF-IDF text vectorizer is used to calculate the word vector. The transformed vector is then used to calculate cosine similarity.

– Similarity using Spacy

We use SpaCy Similarity feature to compute the difference. The word embedding is generated using word2vec model. The resultant is then computed by taking the negation of the similarity.

Most of the above-mentioned projects work best for standard English sentences. However, we created a validation to have a comparison of which model suit our case. We finally concluded that calculating difference using the vector method gave higher accuracy.

The effort estimation is directly proportional to the changes made to the commit change. Thus the effort estimate can be given as

$$\frac{(1 - codesimilarity)}{totalnumberofcharacters}$$

There are other complex methods to compute the similarity of code, like using Abstract Syntax Tree and Measure of Software Similarity (MOSS) (Schleimer et al. [2003]). We leave it for the future to explore.

### *Developer - code connected graph*

Every contributor touches a certain piece of code as a core developer or as a contributor. There is a difference in the pattern for the different types of project. Scientific software like PETSc and FLASH don't have more client specific customizations.

Projects like l2met, couchpotato or famous have a much more single developer specific code. The graphs are generated by creating a directed edge from a developer to a file, to which they have committed changes. Graph measures like density, the in-degree, out-degree, and centrality depicts some important characteristics of the project. We intend to explore this further in the future.

### *Developer - contributor connected graph*

There is a lot of interaction in between team and the frequency of response varies from between the team to a user. Some developers are proactive and have a tendency to respond to users frequently. Whereas some contributors keep it to themselves. There are many such patterns that we can study by just using the developer contributed graph.

The developer contributor graph is a weighted graph with edges between a developer and other members of the team based on communications between developers. The weight is based on the frequency of the developers' responses. At times when the mail chain address reply to all, the edge is connected from the sender to all the receivers. In case of the Git issues, the response is a weight on the issue thread. However, a user may address certain concern using the "@⟨user_name⟩" structure. In such a case, it is strictly dedicated to that specific Git user.

# CHAPTER V

# RESULTS AND ANALYSIS

## Data

We collected data from various sources to perform the software productivity measures and understand its implications. For projects such as PETSc, on an average it takes around a day to download the data. The average number of emails is 20,000. Because most of the data and computation are stored in MongoDB, the size of the collection is about 120 MB. The major challenge was downloading email data for LINUX. The dataset size is around 6.5 GB.

For parsing the Git log files it takes a couple of hours on average depending on the project size. The size of MongoDB collection ranges from 4 MB to 30 MB.

The other challenge is downloading issues from GitHub using the REST API. Git restricts a maximum of 5,000 calls an hour to an authorized user. The size for SPACK issues is 20 MB.

## Topic Labelling

The result of the topic labeling is shown in Table 1. Understanding the bugs and features classification helps determine the stability of the project. Things to consider are how the teams deal with the bugs? What are there project lifecycle like? How much do they value third-party users? Does the community support matters? If so, how? Does the popularity of the project depend on these factors?

TABLE 1. Topic label analysis

| Project | Bug | Feature | Total | Duration (months) |
|---|---|---|---|---|
| PETSc | 2,538 | 18,334 | 20,880 | ∼ 146 |
| FLASH | 588 | 111 | 699 | ∼ 195 |
| LINUX | 79,217 | 5,13,480 | 5,92,697 | ∼ 264 |
| VTK | 36,212 | 35,498 | 71,710 | ∼ 224 |

As we can see for PETSc and Linux, around 12-13% of the discussion is centered around bugs, and the remaining emails are about feature requests or other activities. For VTK, around 50% of the emails are about bugs. In terms of Google searches for each package, the popularity of VTK and PETSc varies a lot. The Google trends in Figure 1 show the normalized counts of searches for both projects over the last five years.

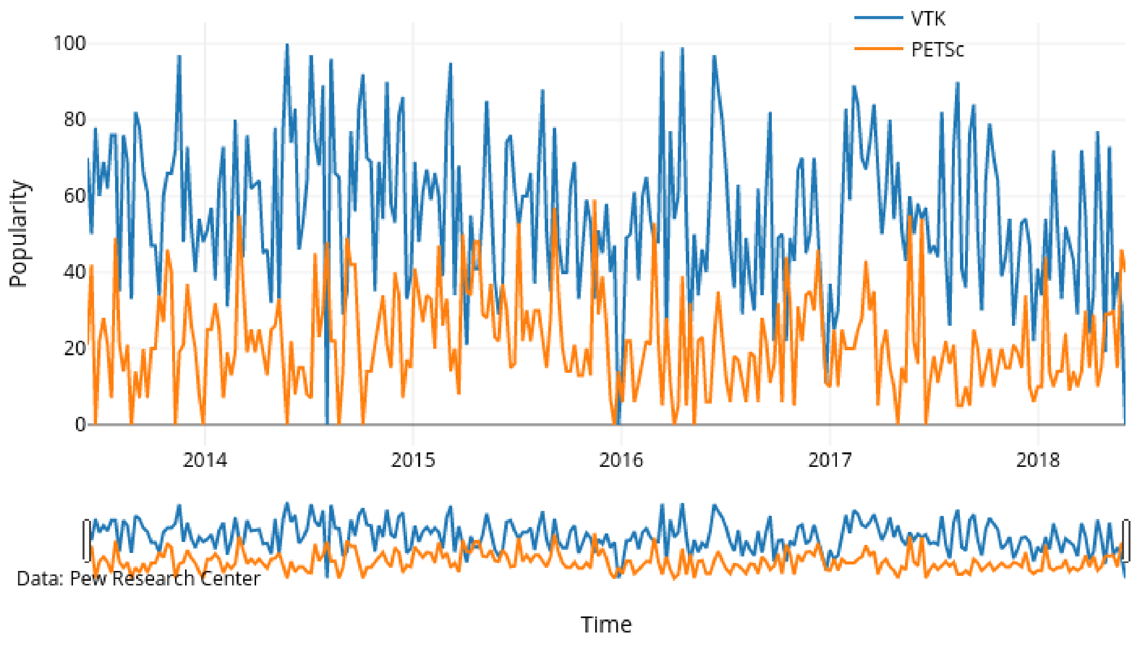
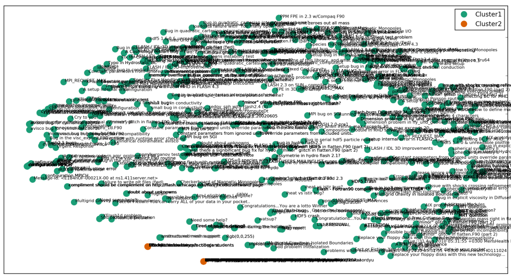
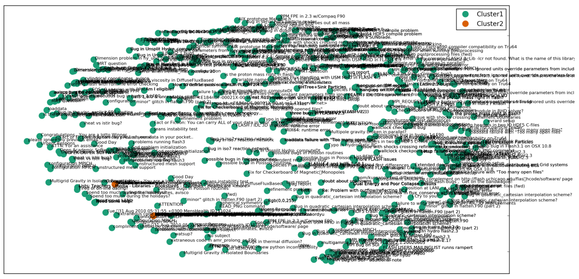


FIGURE 1. Google trends - VTK vs PETSc

For FLASH, the majority of emails were about bugs and a very few were about features. This was a surprisingly different pattern as FLASH team had the



(a) PETSc



(b) FLASH

FIGURE 2. K-Means clustering.

practice to take feature request outside email conversations and it was confirmed by one of the lead team members. Thus we see not many variations in the FLASH dataset. GitHub issues already have tag associated with them, so did not require tag generation through NLP.

## Clustering

One of the goals of using clustering is to help identify anomalies. There are various other features one can consider while executing anomaly detection. We considered the title as the feature to figure out completely off topic discussion. Since the scientific domain are restricted with the domain experts, we detected few outliers. The number of spammers was also limited. A good reason for the same is most mail servers have spam detectors already enabled. The clustering results for PETSc and FLASH can be seen in Figure 2.

As we can see, The two outliners are titled “New web utility” and “Student Insurance”. These two points clearly don’t speak about the working of the software or issues related to it.

## Visualizations

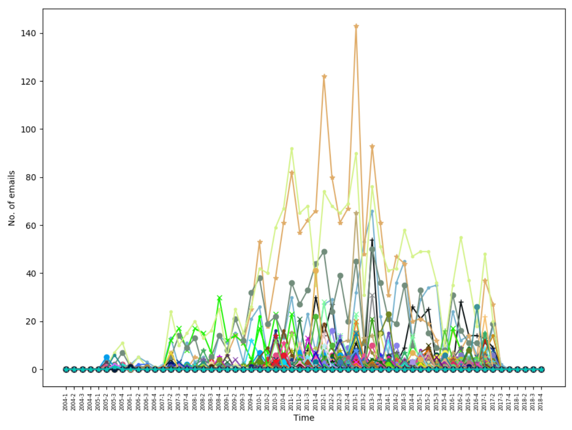
Visualizing developer activity over time can help identify core developers' normal work patterns or abnormal periods (e.g., vacations or other events). This can help predict when bugs/features associated with that developer are likely to be resolved. It could also help with planning new releases. We show these visualizations for PETSc, FLASH, LINUX, VTK, and SPACK in Figure 3.

On the x-axis, we have time (measured in months) and the value on the y-axis is the number of emails a developer exchanged during that month.

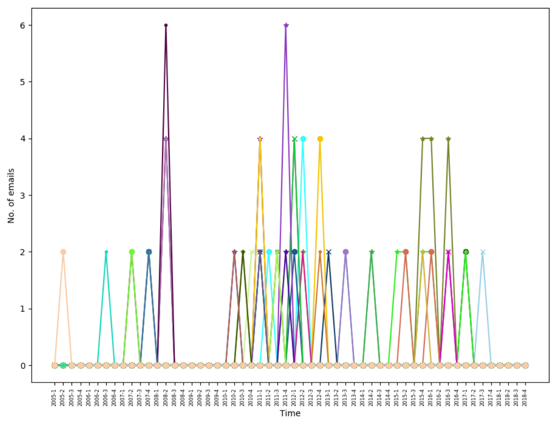
The peaks on each of the graphs show the most active developer. It is very evident the places where those line plots drops are the regular vacation (or non-development) periods for that developer. The density of the lines on the bottom shows the popularity of the project and the lines with different color peak shows the number of active developers.

Understanding how long an email chain runs both in terms of duration and amount together gives interesting insights like the most common recurring topic, or an important concern which has not been resolved since long, or concluding its a complex topic. Further, one can evaluate how many topics are been documented and also provide suggestion to document. The teams can in general work towards resolving issues much quicker. For projects, PETSc, FLASH, VTK, and SPACK the plots are represented in Figure 4.

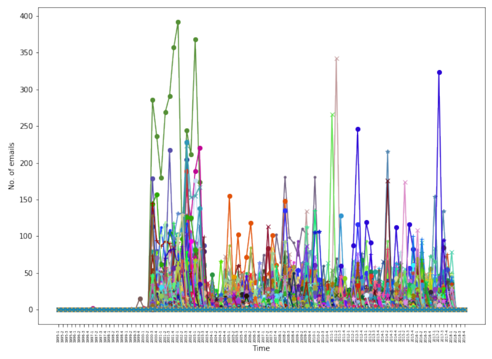
The X-axis represents email topics, Y-axis email count. The diameter of the circle gives time complexity. Thus, we can see for PETSc "Configuration issues" have most time complexity and a lot of emails were exchanged discussing "estimating eigenvalues" in a short span of time.



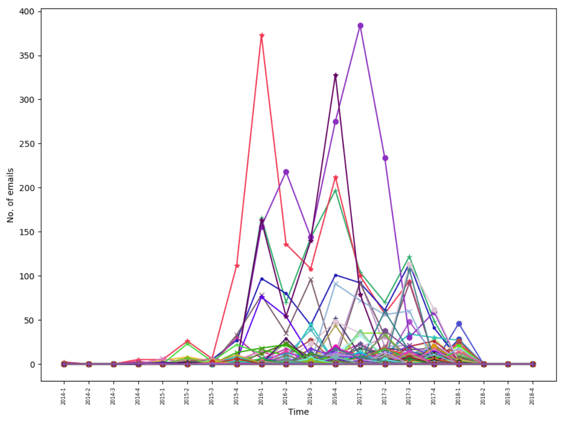
(a) PETSc



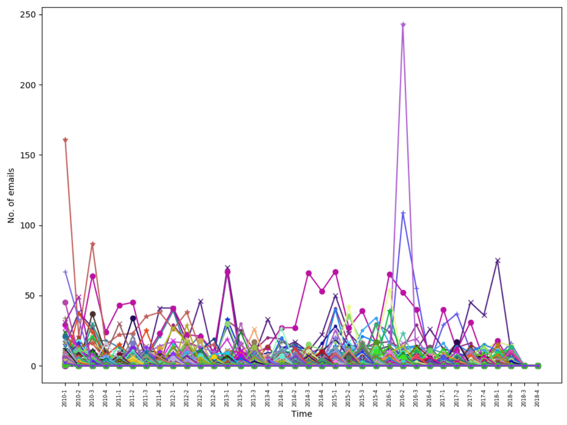
(b) FLASH



(c) LINUX

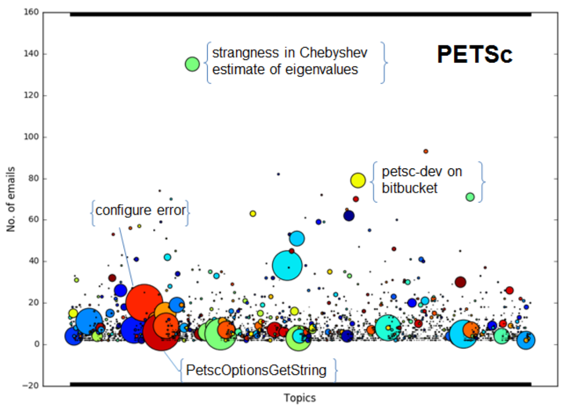


(d) SPACK

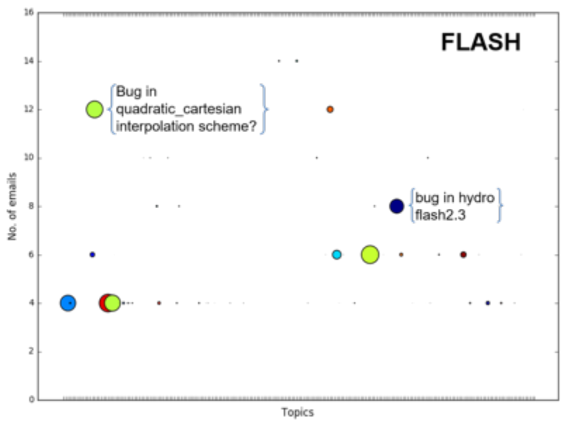


(e) VTK

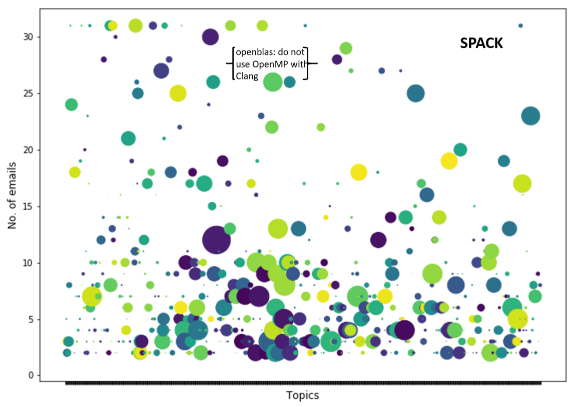
FIGURE 3. Email and issue counts per developer.



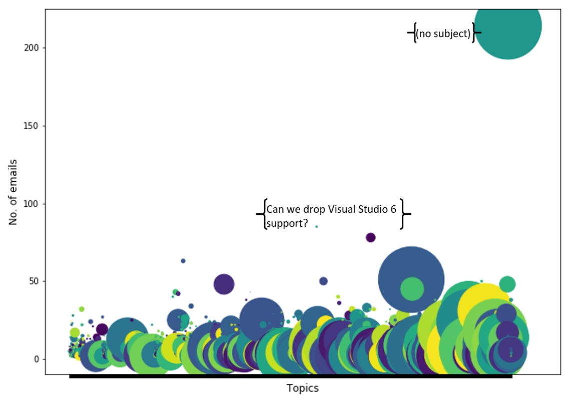
(a) PETSc



(b) FLASH



(c) SPACK



(d) VTK

FIGURE 4. Topics over time and count

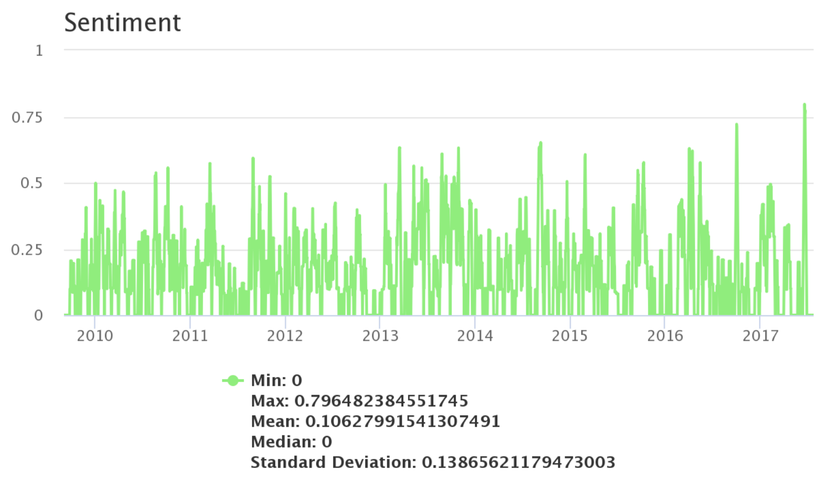
For VTK, the team had a discussion on a title “no subject” for almost 200 emails. Also, a discussion on support for Visual Studio went along for almost 80 emails within a duration of just three days. It is evident the community is very active and most of the discussion involved a lot of stakeholders. SPACK has a default pattern, with a usual discussion not involving more than 40 responses on either of the topics. The reason for such a behavior is probably because of the age of the project. It is just 4-5 years old.

## Sentiment Analysis

Sentiment analysis offers us a glimpse into the emotional state of the team, at least as visible through development-related communications. One can judge the attitude, options, and opinion of the repository. Further, does sentiments affect the team spirit? Can it make or destroy the team spirit? How do the trends follow? How are time variant the projects? Many evaluations that can be done while studying this time-dependent sentiment analysis.

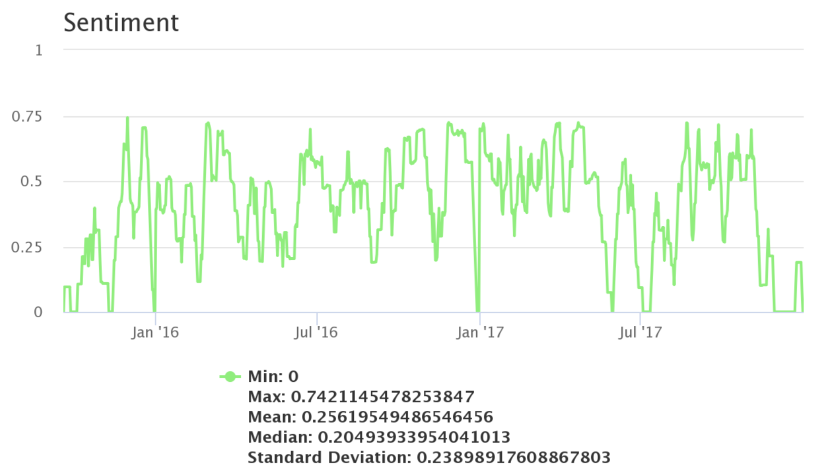
Sentiment analysis for individual developer across different teams is projected in Figure 5.

Heat maps in Figure 6 show the variation of the developer sentiments from the average team mood for a month in a year. In research projects, we see certain patterns especially at the start of a fiscal year when new funding starts and at the end when there is an uncertainty of the budgets for next year. The heat maps are for the time period when the team had maximum amounts of conversation happening. Most of the teams’ sentiment is below average during the middle of the year in general.



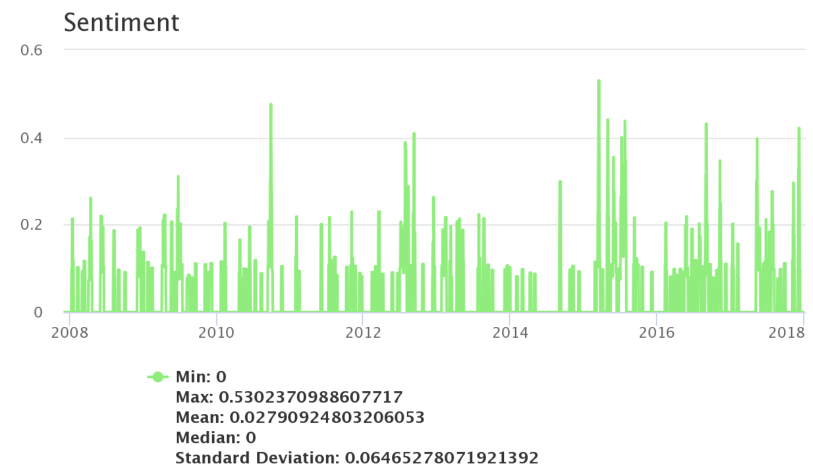
Min: 0
Max: 0.796482384551745
Mean: 0.10627991541307491
Median: 0
Standard Deviation: 0.13865621179473003

(a) PETSc



Min: 0
Max: 0.7421145478253847
Mean: 0.25619549486546456
Median: 0.20493933954041013
Standard Deviation: 0.23898917608867803

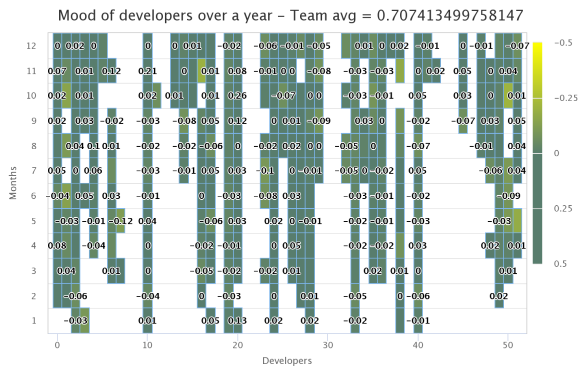
(b) SPACK



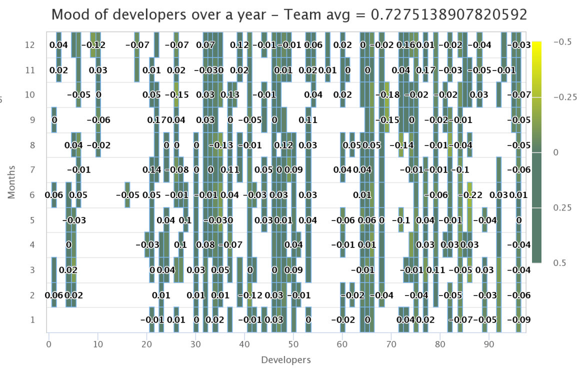
Min: 0
Max: 0.5302370988607717
Mean: 0.02790924803206053
Median: 0
Standard Deviation: 0.06465278071921392

(c) VTK

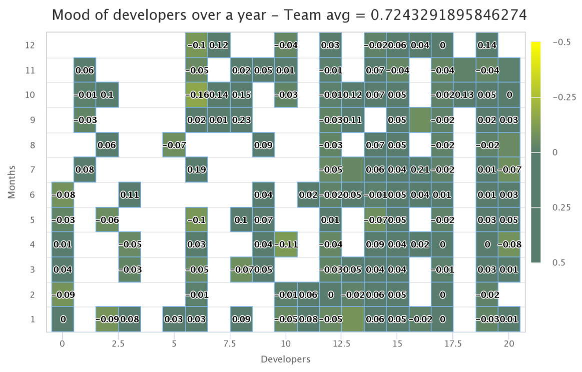
FIGURE 5. Developer sentiments over time.



(a) SPACK (2016-17)



(b) VTK (2009-10)



(c) PETSc (2014-15)

FIGURE 6. Team sentiments over time.

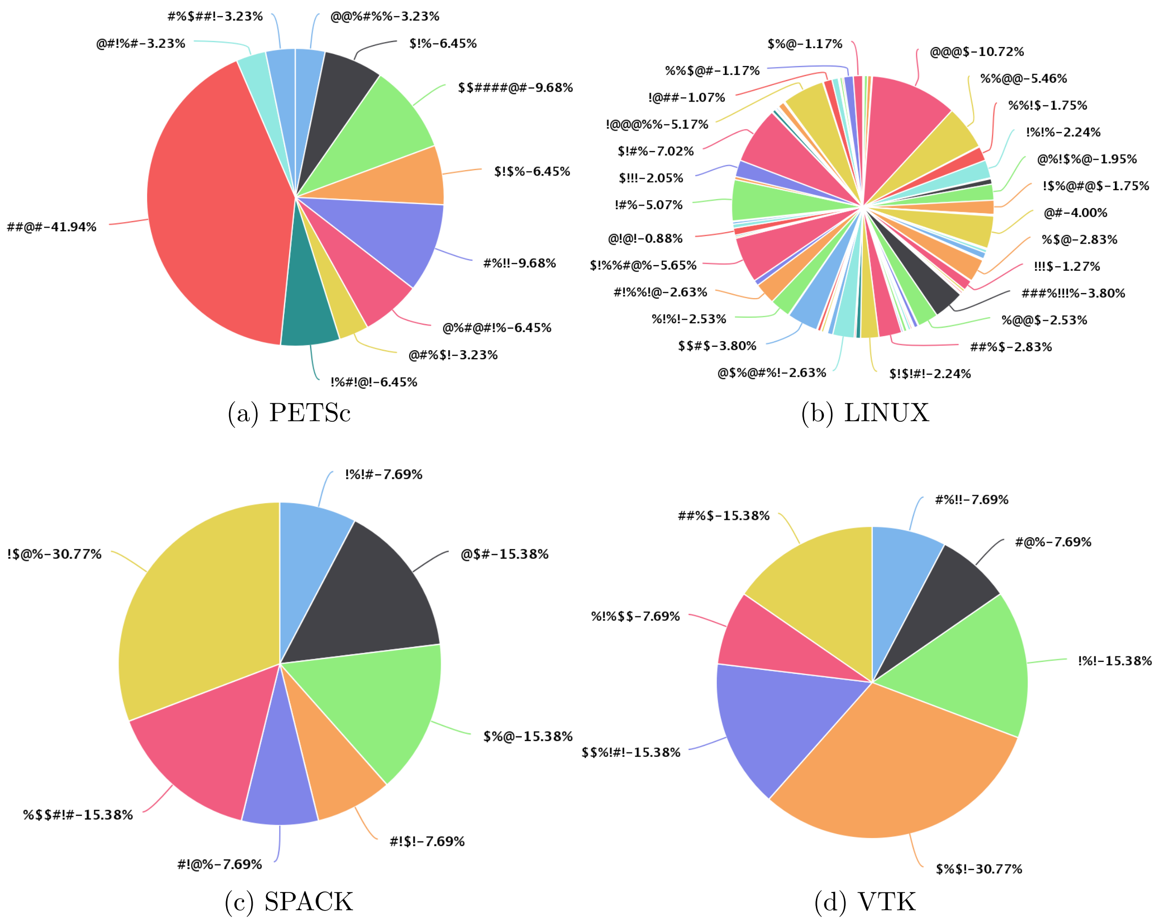
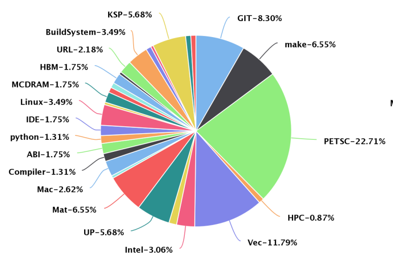


FIGURE 7. Profane words.

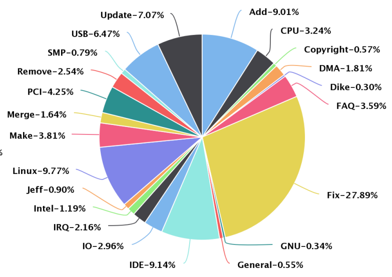
## Profanity

Every community is different. The use of curse word differs from one project to another. Is the use of such words specific to a region? Or is there a specific pattern of usage? Whats the context of such usage? We first explore the variety and number of curse words used with the pie charts for PETSc, SPACK, VTK, LINUX in Figure 7.

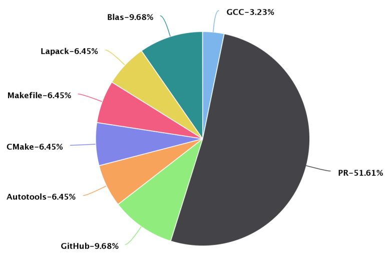
Although the community size differs for each of the projects, the scientific software shows pretty much the same words. For Linux, the number of contributors



(a) PETSc



(b) LINUX



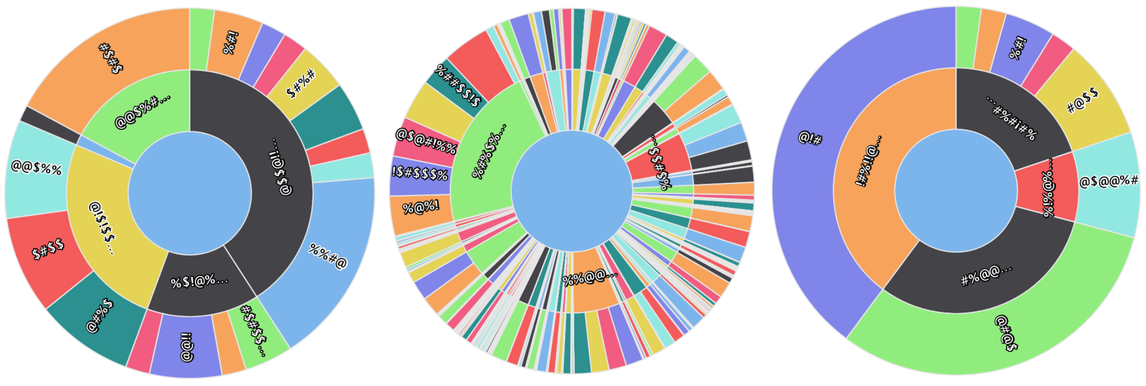
(c) SPACK

FIGURE 8. Topics corresponding to profanity.

is much higher, we see a pattern where even the females curse, unlike any other project we've analyzed. The curse words used are also more varied.

An interesting insight into profanity is related to understanding what topics most commonly inspire it. For example, managing multi-platform configuration and build systems is a time-consuming and unpleasant task for scientific software developers. Indeed, the pie chart reveals that some of most common topics associated with profane words are configuration and build-related. The pie charts for PETSc, SPACK, and LINUX are shown in Figure 8.

The contributors are also different in all the above-mentioned projects. Not every developer curses or the curse word in one mail chain may be used in different ways from other conversations. An interesting aspect to explore would be the usage of gender-biased words. In this study we considered individual projects separately, but we could also study the same developers across all their open-source projects. Much interesting information can be observed by these sunburst charts for PETSc, SPACK, LINUX that are represented in Figure 9. We can easily see the most favorite curse word for each of the developers.



(a) PETSc (b) LINUX (c) SPACK

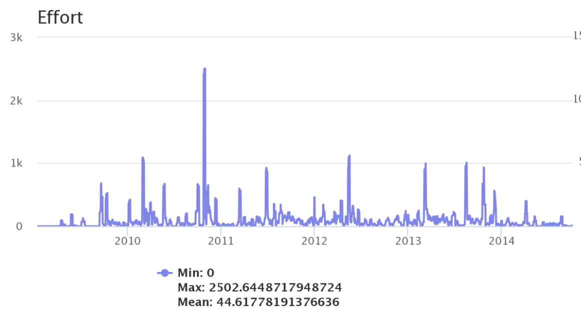
FIGURE 9. Developers' usage of profanity. The inner ring represents the developer's name and the outer ring is the profane word used by each. We have encoded both the developers' names and the curse words for publication purposes.

## Estimating Developer Effort

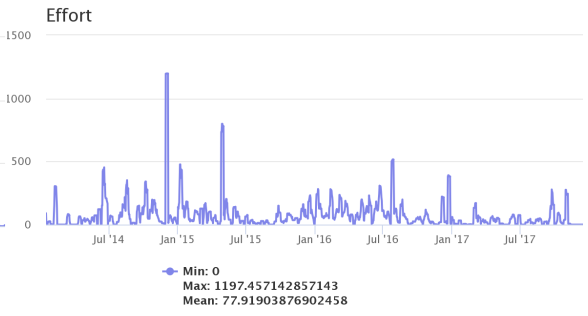
Many measures have been defined in an attempt to quantify software developer efforts, such as the number of commits, days of activity, number of changed files or lines of code, the frequency of commits, etc. All these measures may work fine in the traditional project environment. For scientific software, the stakeholders are different. Developers are from different domains and have a different level of expertise. Thus, we decided to average out the development effort of a single developer in a team and analyze it.

The plots in Figure 10 represents the moving average of one of the top developers in each team at a certain time range. The little peaks we see in the plots are probably for the time when the projects are having alpha releases.

As mentioned earlier, the sentiments may have some correlation with the development effort. Can the positive correlation stating the negative a person goes, more likely he/she loses interest in the project? Or a contributor started being

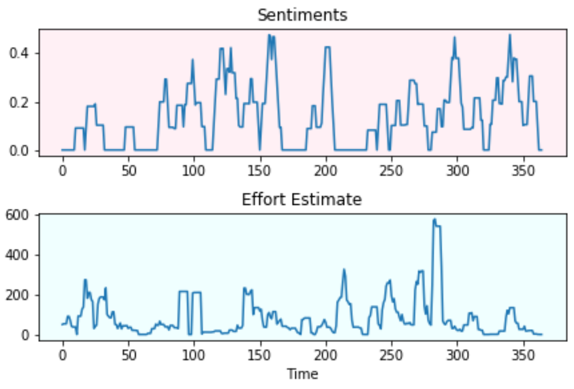


(a) PETSc

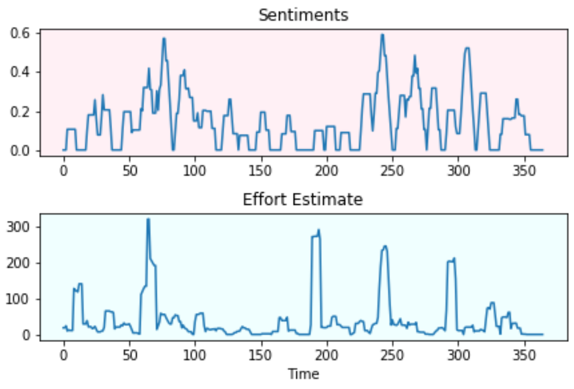


(b) SPACK

FIGURE 10. Individual developer effort estimate.



(a) PETSc



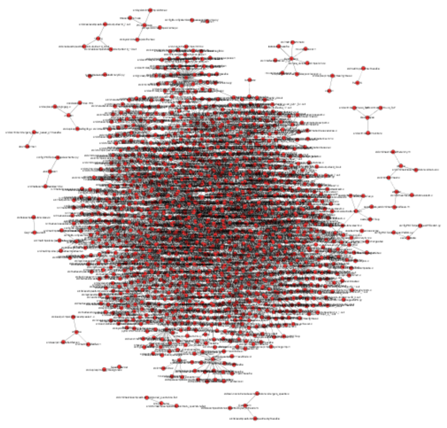
(b) VTK

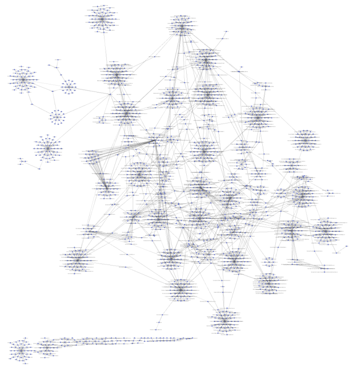
FIGURE 11. Developer sentiment vs effort.

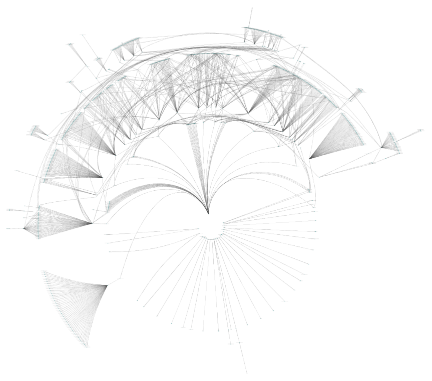
frustrated with some functionality and eventually end up resolving the issue and started liking the project. Do high motivated people influence other on the team? It is interesting in finding a correlation between the two. The plots in Figure 11 show these correlations.

## Developer - Code Graph

Every contributor touches a certain piece of code as a core developer or as a contributor. There is a difference in the patterns for the different types of projects. Scientific software projects like PETSc and FLASH don't have specific


(a) PETSc


(b) VTK


(c) SPACK

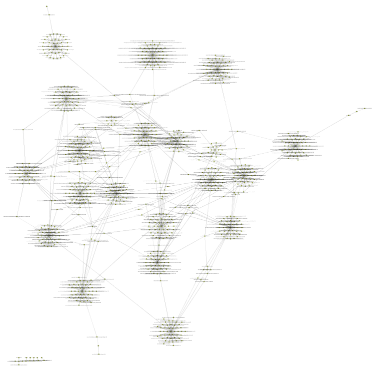

(d) FLASH

FIGURE 12. Developers to code relations (scientific software).

customizations for specific users, whereas projects like CouchPotato contain much more customer-specific code. The graphs are generated by creating a directed edge from a developer to a file they commit. Graph measures such as density, in-degree, out-degree, and centrality depict some important characteristics of the project. We intend to explore more graph analysis techniques in the near future.

Table 2 projects the maximum topological coefficient and eccentricity of the developer to code graph representation. Topological coefficient is a relative measure for the extent to which a node shares neighbors with other nodes. Eccentricity is the maximum distance between a vertex to all other vertices. Just with these two measures, we can conclude that the graphs are densely connected. We will explore more such metrics in near future.

| Project | Maximum Topological Coefficient | Maximum Eccentricity |
|---|---|---|
| PETSc | 0.5834 | 12 |
| VTK | 0.526 | 14 |
| SPACK | 0.714 | 12 |
| FLASH | 0.53 | 11 |

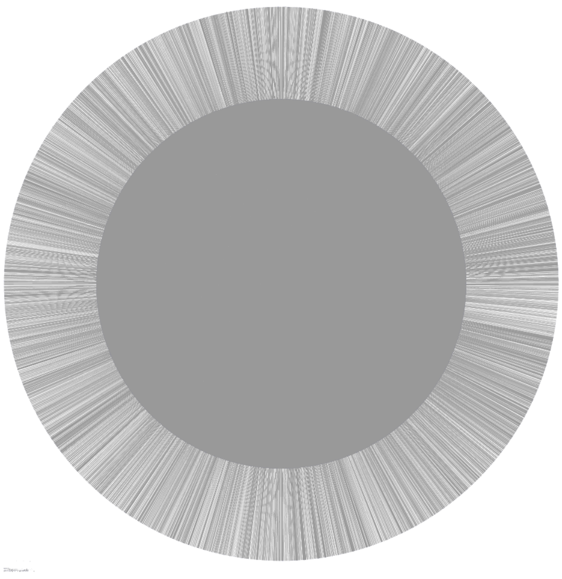
TABLE 2. Graph measures for developer to code graph.

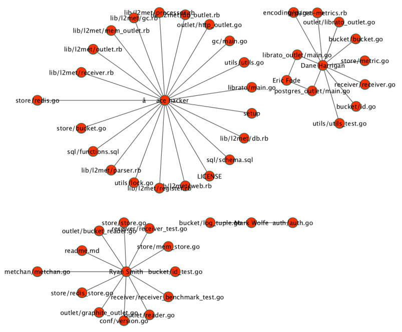
For scientific software the graphs are shown in Figure 12 and for more open source projects in Figure  13.

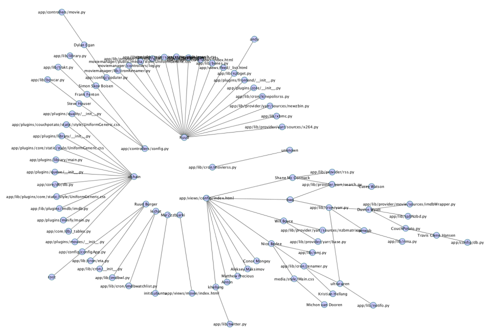
We observed that all scientific software had at most five developers who are more active than most of the other developers, which is far fewer than in typical product-oriented industry software projects.

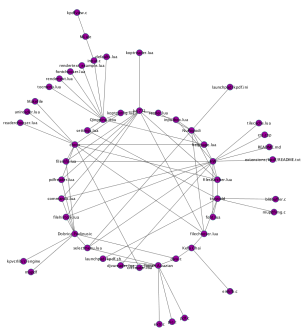
## Developer - Contributor Graph

Some humans are proactive, some are reactive while some are extreme introverts. It is interesting to study the influencer in the team. We plotted the


(a) LINUX


(b) l2met


(c) COUCHPOTATO


(d) Kindle2Pdf

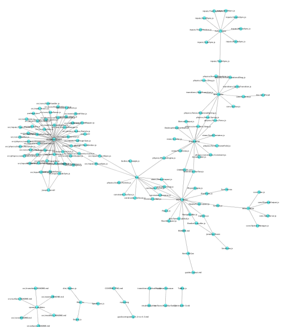
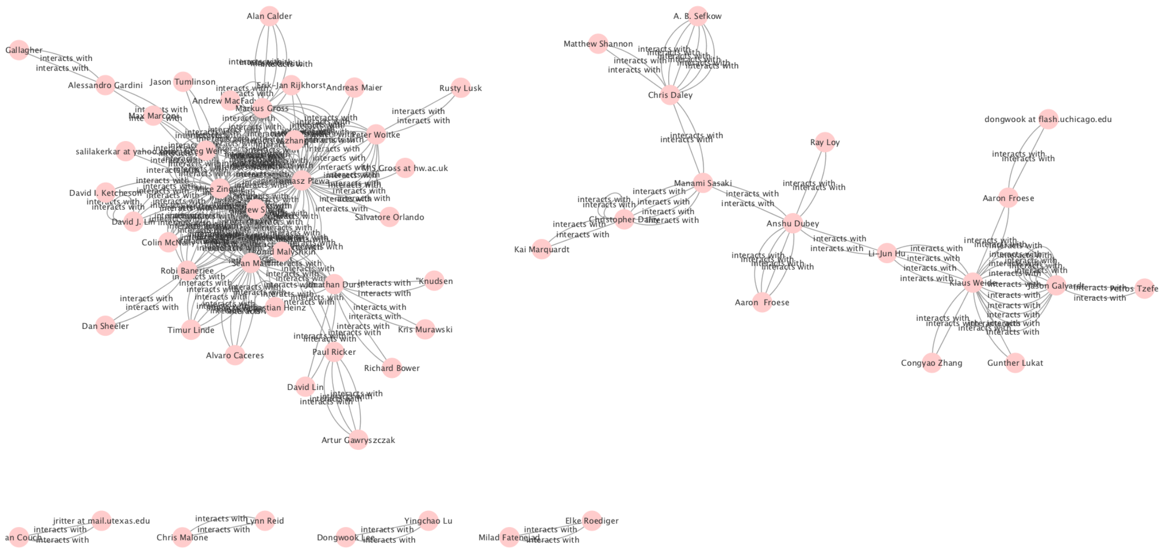

(e) FAMOUS

FIGURE 13. Developer to code relations.



(a) FLASH

FIGURE 14. Developer to developer relation

developer to contributor graph, weighing the edge on a number of emails exchanged between them. Some developers are more keen on interacting among themselves, while some take charge to respond to users. Does this indicate a team strategy or is it just human nature? Can someone be gender-biased and not respond to others selectively? Does not responding impact the users? We studied these aspects for both the traditional projects and scientific projects.

Figure 14 shows the developer-developer graph for FLASH. In most of the team, there is only one or two developers who respond to end users, others usually interact among themselves. The out-degree centrality shows the same. We also observe completely disconnected subgraphs, indicating small groups of people who communicate among themselves but not with the overall team.

| Developers | Self Loops | Partner of Multi-Edged Node Pair |
|---|---|---|
| Developer1 | 686 | 143 |
| Developer2 | 259 | 124 |
| Developer3 | 870 | 112 |
| Developer4 | 396 | 89 |

TABLE 3. Graph metrics for PETSc developer to contributors graph.

Table 3 represents some metrics for PETSc graph generated by creating directed graph from one developer to another (the developer names are omitted for privacy reasons). It is difficult to visualize the graph statically, so we have listed these measures. We observe that some developers use "respond to all" more frequently and hence have a lot more self loops.

## Miscellaneous

We also tried to perform politeness analysis in order to get correlation with profanity, but the results show that there is no strong negative or positive correlation between them. We also did interrogation analysis where we wished to isolate the questions asked in the email conversation. This could help us evaluate frequently asked questions, which could be documented. We were able to figure out some of the questions asked, for example:

– Have you tried with the master branch?

– Do either of these not redistribute if asked?

– Can Cubit produce MED or be converted to MED?

– Hi Developers, Do we increase the patch number when pushing a new patch into maint?

  - Is that a 3rd test run in addition to the above two?

Some of them were pretty much obvious questions, which a domain expert can answer without knowing the context. The not obvious ones led to extended conversation in the email chain. Thus, to understand the context we need to perform anaphora (pronoun) resolution. In order to do so, we used the techniques we had mentioned in section 2.12. However, both the libraries work best when the sentences are grammatically correct. Since we are dealing with emails, the user is free to express in any form they want. Thus, we have left this line of investigation for further research.

# CHAPTER VI

# CONCLUSION

A lot has been explored, yet there is much more we can do. We have seen that the scientific software community is different than traditional software development teams. We have tried to explore all sorts of repositories for this work. For example, PETSc which is an older project with the average developer community size, SPACK which is relatively new yet mature, Linux which is probably the oldest open source and extremely large project, and small repositories on git which are archived (not active). Some at least in part rely on government funding, while others are not explicitly funded at all. Every repository has their ups and downs. We have tried to explore projects using different metrics that vary over time and correlations between some of these metrics.

Much of the productivity-relevant data in scientific software development projects is in the form of emails. We have used NLP techniques to estimate productivity metrics reflecting developer effort. We were able to auto-tag emails as bugs or features and identify important information. The classification of emails gave us a fair idea about what topics of discussions every project follows and a rate at which each discussion ends. It could increase the productivity if the teams had their issues and workarounds documented for future reference.

Since the unstructured dataset can give us so much information, imagine the possibility if we have structured data. It is worth considering using organized and precise emails and commits messages. This study also shows a use of issue trackers is beneficial and we hope to motivate project teams to use them.

Understanding the human behavior in terms of sentiment, profanity or politeness helps understand team morale and identify topics that may have negative impact on it or productivity. We proposed our own effort estimation metrics which are not as complex as other existing standard effort estimation metrics and do not require constant developer monitoring. Correlating these metrics with other helped us identify a human pattern and may help predict behavior in future research. There are a lot of factors such as gender, region, experience, age, highest earned degree etc that are yet to be considered.

We have looked at various projects from a broader perspective. A comprehensive study of developers with respect to team interaction and file commits have given us patterns for both successful and archived projects. It is clear that scientific software projects work as closed group communities. In future, we plan to study all the repositories on git and apply the patterns we recognize to estimate projects' health.

We intend to explore more complex metrics, such as the rate of bug fixes, and combine them with some traditional metrics. We plan to approach profanity from a wider angle, i.e to understand how certain developers' influence spreads. Some of these studies involve psychological analysis and may require collaboration with psychologists or social scientists.

We have established some ground metrics for future opportunities. The next step would be to further analyze larger numbers of repositories. We plan to build a predictive model based on this work that can help stakeholder monitor and understand different aspects of the health of a project.

Another key aspect is enhancing software quality be it through software testing, documentation, version control, building and installation, and good design as part of improving overall scientific productivity. We hope this work motivates people to conduct research not only for traditional projects but also for research software projects.

# REFERENCES CITED

Ideas Productivity. ideas. `https://ideas-productivity.org/`, 2011. Accessed on 2017-08-1.

Linus Torvalds. Linux. `https://github.com/torvalds/linux`, 1999. Accessed on 2018-03-17.

Ruud Burger. Couchpotato. `https://github.com/CouchPotato/CouchPotatoV1`, 2010. Accessed on 2018-03-17.

David Valdman. famous. `https://github.com/Famous/famous`, 2014. Accessed on 2018-03-17.

Ryan Smith. l2met. `https://github.com/ryandotsmith/l2met`, 2012. Accessed on 2018-03-17.

Qingping Hou and Tigran Aivazian. kindlepdfviewer. `https://github.com/koreader/kindlepdfviewer`, 2011. Accessed on 2018-03-17.

Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2018. URL `http://www.mcs.anl.gov/petsc`.

Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. The spack package manager: Bringing order to hpc software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 40:1–40:12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3723-6. doi: 10.1145/2807591.2807623. URL `http://doi.acm.org/10.1145/2807591.2807623`.

Flash Center at University of Chicago. flash. `http://flash.uchicago.edu/site/`, 2011. Accessed on 2018-03-17.

Will Schroeder, 1968 Martin, Ken, Bill Lorensen, and Inc Kitware. *The visualization toolkit : an object-oriented approach to 3D graphics.* [Clifton Park, N.Y.] : Kitware, 4th ed edition, 2006. ISBN 9781930934191. CD-ROM inside back cover.

Vineeth G. Nair. *Getting Started with Beautiful Soup*. Packt Publishing, 2014. ISBN 1783289554, 9781783289554.

Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118108.1118117. URL `https://doi.org/10.3115/1118108.1118117`.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. Treebank-3, 1999. URL `https://catalog.ldc.upenn.edu/LDC99T42`.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. *The Penn Treebank: An Overview*, pages 5–22. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-010-0201-1. doi: 10.1007/978-94-010-0201-1_1. URL `https://doi.org/10.1007/978-94-010-0201-1_1`.

M. F. Porter. An algorithm for suffix stripping. In Karen Sparck Jones and Peter Willett, editors, *Readings in Information Retrieval*, chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-454-5. URL `http://dl.acm.org/citation.cfm?id=275537.275705`.

Jason Brownlee. A gentle introduction to the bag-of-words model. `https://machinelearningmastery.com/gentle-introduction-bag-words-model/`, 2017. Accessed on 2018-01-7.

Wikipedia contributors. Latent dirichlet allocation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Latent_Dirichlet_allocation&oldid=844981860`, 2018a. [Online; accessed 18-Feb-2018].

Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

Edwin Chen. Introduction to latent dirichlet allocation. `http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/`, 2011. [Online; accessed 14-Jan-2018].

Wikipedia contributors. Tfidf — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=841608474`, 2018b. [Online; accessed 11-Jan-2018].

Percy Liang. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408, 2013. URL `http://arxiv.org/abs/1309.4408`.

Kevin Clark and Christopher D. Manning. Deep reinforcement learning for mention-ranking coreference models. *CoRR*, abs/1609.08667, 2016. URL `http://arxiv.org/abs/1609.08667`.

Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, and Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction network bib. `http://www.cytoscape.org`, 2003. [Online; accessed 10-June-2018].

Martin Fowler. Cannot measure productivity. `https://martinfowler.com/bliki/CannotMeasureProductivity.html`, 2003. Accessed on 2017-04-27.

Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: Evaluating contributions through discussion in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 144–154, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3056-5. doi: 10.1145/2635868.2635882. URL `http://doi.acm.org/10.1145/2635868.2635882`.

Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3056-5. doi: 10.1145/2635868.2635922. URL `http://doi.acm.org/10.1145/2635868.2635922`.

Ferdian Thung, Tegawende F. Bissyande, David Lo, and Lingxiao Jiang. Network structure of social coding in github. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 323–326, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4948-4. doi: 10.1109/CSMR.2013.41. URL `http://dx.doi.org/10.1109/CSMR.2013.41`.

Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi: 10.1145/2597073.2597118. URL `http://doi.acm.org/10.1145/2597073.2597118`.

Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Perceptions of diversity on github: A user survey. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '15, pages 50–56, Piscataway, NJ, USA, 2015a. IEEE Press. URL `http://dl.acm.org/citation.cfm?id=2819321.2819330`.

Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark G.J. van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3789–3798, New York, NY, USA, 2015b. ACM. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702549. URL `http://doi.acm.org/10.1145/2702123.2702549`.

Oskar Jarczyk, Błażej Gruszka, Szymon Jaroszewicz, Leszek Bukowski, and Adam Wierzbicki. Github projects. quality analysis of open-source software. In Luca Maria Aiello and Daniel McFarland, editors, *Social Informatics*, pages 80–94, Cham, 2014. Springer International Publishing. ISBN 978-3-319-13734-6.

Stephen H. Kan. *Metrics and Models in Software Quality Engineering.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. ISBN 0201729156.

Barry Warsaw, Mark Sapiro, and Abhilash Raj. A gentle introduction to the bag-of-words model. `http://www.list.org`, 1999. Accessed on 2017-06-24.

Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkely, CA, USA, 2nd edition, 2014. ISBN 1484200772, 9781484200773.

GitHub Developer. Github developer api v3. `https://developer.github.com/v3/`, 2017. Accessed on 2018-03-17.

Scott Chacon and Jason Long. 9.2 git and other systems - migrating to git. `https://git-scm.com/book/en/v2/Git-and-Other-Systems-Migrating-to-Git`, 2005. Accessed on 2018-01-12.

Richard Stephens. Swearing is actually a sign of more intelligence - not less - say scientists. `https://www.sciencealert.com/swearing-is-a-sign-of-more-intelligence-not-less-say-scientists`, 2017. Accessed on 2018-02-8.

Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 76–85, New York, NY, USA, 2003. ACM. ISBN 1-58113-634-X. doi: 10.1145/872757.872770. URL `http://doi.acm.org/10.1145/872757.872770`.