Using Topic Models to Understand the Evolution of a Software Ecosystem

Nicolas Lopez
Department of Informatics
University of California, Irvine
Irvine, CA 92627-3440 U.S.A.
nlopezgi@uci.edu

ABSTRACT

The development of a software system is now ever more frequently a part of a larger development effort, including multiple software systems that co-exist in the same environment: a software ecosystem. Though most studies of the evolution of software have focused on a single software system, there is much that we can learn from the analysis of a set of interrelated systems. Topic modeling techniques show promise for mining the data stored in software repositories to understand the evolution of a system. In my research I seek to explore how topic modeling techniques can aid in understanding the evolution of a software ecosystem. The results of this research have the potential to improve how topic modeling techniques are used to predict, plan, and understand the evolution of software, and will inform the design of tools that support software engineering activities such as feature location, expertise identification, and bug detection.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring, reverse engineering, and reengineering*

General Terms

Design, Experimentation, Measurement, Theory

Keywords

Mining software repositories, software evolution, software ecosystems, topic modeling, Latent Dirichlet Allocation

1. INTRODUCTION

Most modern software systems are no longer developed in isolation. Instead, they are composed of large sets of interrelated libraries, applications, and frameworks, each of which is usually developed and maintained by a separate team [5]. These complex environments that span multiple systems, developers, and organizations are commonly referred to as software ecosystems [8]. Analyzing the evolution of a software ecosystem can enable the identification of global properties of the code such as reuse patterns [16], provide new insights into their organization and governance [7], and shed light on the complexity of current software development practices [5].

Topic modeling techniques have emerged as a promising approach to understand the evolution of a software system [10, 18]. A particularly popular class of techniques, collectively referred to as LDA, uses machine learning algorithms to produce Latent Dirichlet Allocation models [3]. Taking as input a code base, these algorithms generate a set of *topics*, each a collection of words that commonly show up together in the code.

LDA models have been used to analyze a single version of a code base [2, 13, 14]. However, these techniques had to be extended to enable the analysis of the evolution of software. Particularly, the development of topic evolution models have enabled analysis of an evolving document corpus [15, 18]. To date, the application of topic evolution models in the case of software has fallen short in two ways: (1) it has relied on models in which the topics themselves do not evolve, limiting analysis of the system to a constant set of terms for each topic, and (2) the analysis has been limited to a very small set of unrelated systems [10, 18], not an ecosystem. This proposal contributes a new strategy to create *fine-grained topic evolution models* in which topics can evolve. Furthermore, it stands out in its use of topic evolution models to explore the evolution of a software ecosystem.

The main goals of this research are: (1) to contribute to the open discussion on how to use topic modeling to understand the evolution of software; (2) to identify trends and patterns in the evolution of topics that span the boundaries of a single software system; (3) to identify interesting phenomena regarding how developers impact the evolution of the topics of a software ecosystem; and, (4) to provide insight that informs the design and development of tools that leverage topic modeling, such as tools that support feature location, expertise identification and bug detection.

1.1 Research Question

The main research question explored in this proposal is as follows:

How can topic models be leveraged to provide insight into the evolution of a software ecosystem?

In order to explore this research question, I plan on performing a series of studies that aim at observing specific aspects of the evolution of the code, for a set of interrelated systems that are part of a software ecosystem, from the perspective of the topics located using LDA. The first study aims at identifying topic evolution models that can characterize at a fine grain how the topics for a given software system have evolved over time. To do this, it is necessary to generate different topic models for each version of the code. The key challenge, then, is to identify links between the topics, particularly because of the non-deterministic nature of the LDA algorithms [1].

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2237-9/13/08...$15.00
http://dx.doi.org/10.1145/2491411.2492402

The second study takes a set of relevant projects from a software ecosystem and applies our fine-grained topic evolution model to look at issues regarding the evolution of the topics. The objective is to observe how topics manifest across the software ecosystem over time. For instance, are certain topics well modularized? As another example, are there similarities between some crosscutting concerns across several projects? The third study, rather than looking at the code and topics in isolation, broadens our analysis to consider how developers have worked with the topics. For instance, are there similarities among the topics that a developer works on across several projects? Are there differences between the topics that core developers work on and the topics that less experienced developers work on?

2. BACKGROUND

Latent Dirichlet Allocation (LDA) is a generative statistical model that allows for inference of topics, with the objective of optimizing the distribution of the topics found over the set of documents from which they were derived [3]. The result of applying LDA to a set of documents is a model that defines a set of topics, as well as an associated probability matrix that maps those topics to the source documents. Each topic is defined as a set of terms, with the terms ordered high-to-low in their probability of co-occurring. Using the probability matrix, relationships among the documents can be identified and measured, according to the extent to which two or more documents have high probabilities associated to the same topics.

LDA was developed for natural language corpora, but has gained traction as a promising technique in software. For instance, LDA has been used to assist developers in diverse activities such as code comprehension [14], bug localization [13], feature location [2], and expertise identification [11].

Figure 1 shows an example of a subset of topics for a hypothetical software system. Each topic is defined by several terms; their size indicates their relative relevance. Topics in an LDA model can be manually labeled with the assistance of a developer with appropriate domain expertise [14]. The topics in Figure 1, for instance, could be labeled as being related to *versioning* (orange), *GUI* (green) and *file management* (pink), respectively.

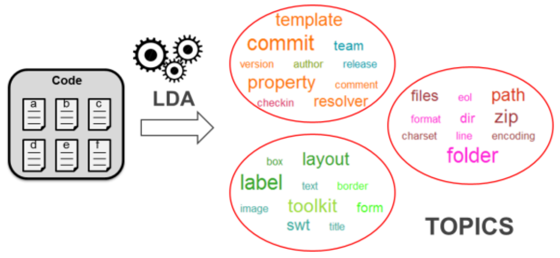

Figure 1. Example set of topics and terms.

The topics produced by LDA can be used to improve comprehension and assist developers in several ways. For instance, Joe, an experienced developer, is performing some maintenance activities. A development tool suggests Joe look at the *file management* topic of Figure 1 since it is related to the files he is currently modifying. The tool shows him other files that are related to this topic. Joe notices there is another file that is also impacted by the current change which he had missed.

As a second example, consider Jill, a developer exploring some code she is unfamiliar with in order to find parts to reuse in a new project. She finds that the *versioning* topic of Figure 1 might be related to one of the features she needs to build in her new system. She decides to explore some classes related to this topic and finds some snippets of code to reuse.

3. PRELIMINARY RESEARCH

As an initial step to explore the usefulness of topics in understanding the evolution of software, we set out to explore whether LDA could be valuable from the perspective of *changing* a system [12]. That is, while most previous research focused on using topics to characterize classes (or methods, or modules), our objective was to find out if they could also be used to characterize the set of classes being changed as part of a single commit.

To find out if topics characterize changes, we performed a study analyzing the change history of an open source system, Mylyn. In this study, we measured the extent to which the classes changed together as part of a commit can be characterized by a small subset of the topics. We defined metrics for this characterization and compared values for the actual changes (the set of classes included in each commit) with equivalent sets of randomly selected classes.

Results from our study demonstrate that topics indeed characterize changes. Figure 2 compares values of our metric ($M(TC_m, TM_{n\alpha\beta})$), that quantifies the extent to which topics characterize a set of classes, corresponding to over 3000 commits (blue) versus random sets of classes (red). The *x* axis corresponds to variations in the number of classes included as part of a commit/random set. As the figure indicates, our metric results in higher values for commit data regardless of the size of the set.

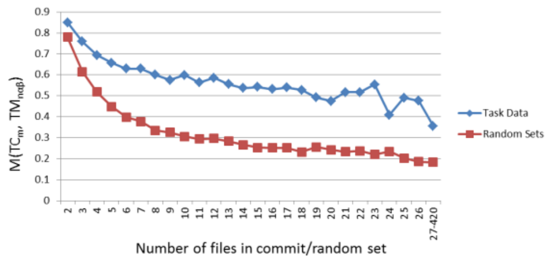

Figure 2. Match between topics and files committed together (blue) and topics and random sets of files (red) with varying size of the set (x axis)

In addition, we made other findings regarding the application of LDA to software. We found that: (1) having access to additional information regarding how developers made the changes, such as which files they opened and examined, but did not actually modify, did not have an impact on the results; (2) the results of the algorithm were highly sensitive to the values of the configuration parameters of LDA; and (3), the settings recommended for topic models when applied to NL document corpora [6] yielded very sub-optimal results.

This study represents an important stepping stone in our research. As we look at the evolution of software, we need to look at the changes. This study provides evidence of the potential of topic location for the analysis of the evolution of software, as LDA can be useful beyond capturing a static version of the topics for a snapshot of the code.

4. PLANNED WORK

This section introduces a series of studies that we are planning to embark upon to address the main research question.

4.1 Fine-grained Topic Evolution Models

Now that we have established that topics are also represented in the changes that developers make, we can go one step further to look at the evolution of the topics for a software ecosystem. The key challenge is that LDA algorithms are non-deterministic [1], which means that running topic location with different versions of the code (in fact, even with the same version of the code), may result in a different set of topics.

This can be problematic if we consider, for instance, what happens if we look at two versions of the same file at different points in time, and at the changes that impacted this file to get from one version to the other. Even though the topics that characterize the different versions of the file might be similar (they might have some matching terms), they are most likely not going to be exactly the same. The challenge is that it is difficult to decide if the variations in the topics are the result of the changes made to the file, or if they are caused by the non-deterministic nature of the LDA algorithm. Moreover, even if the results were deterministic, what level of difference indicates really different topics versus topics that simply evolved somewhat? We need to define an approach to create equivalences among the topics corresponding to the different versions of the code.

In this first study, we will identify *fine-grained topic evolution models* that characterize how each topic has progressively changed as a result of changes in the code. Our approach differs from previous research regarding the use of topic evolution models for software [10, 18] in several ways: (1) we extend the Link topic evolution model, originally devised for NL document corpora [15], for use on code; (2) we build on recent research regarding the stability of topics in LDA models [1]; and (3), we bring this previous research together with the approach from our previous study, indicating how changes are related to topics.

To create our fine-grained topic evolution models we need to identify topics for each consecutive version of the code and then establish links between these topics. To deal with the non-determinism of LDA, we will run the algorithm multiple times on each version of the code. This will allow us to identify stable topics produced by the algorithm and reduce the impact of topics that show up randomly. To include the impact of changes on the topics found version to version, we will quantify the extent to which a topic should change according to the topics that characterize the changes made in the code, and use this information to fine-tune how we create links between the topics.

To evaluate our fine-grained models we will run a comparative study of the performance of our topic evolution model versus previous topic evolution models [18]. We will quantify their performance by using as gold set the evolution of a system previously studied in [17].

4.2 Finding Patterns in Topic Evolution Models

Using the results from the previous study, we can now look at a software ecosystem from the perspective of topics and change. Our second study will be an exploratory study, one in which what we find initially will shape what we look at in more depth. We will start by opportunistically selecting a subset of projects that are part of a software ecosystem, and analyzing them using our fine-grained topic evolution model described in Section 4.1.

In this study, we are particularly interested in observing patterns related to topics, in an analogous way to how CodeCity [19] was used to find patterns in the evolution of software in terms of classes and packages (God class, stable/unstable packages, functionality that moves around constantly, etc.). That is, we will use our lens of changes and topics to design and build visualizations, and through the visualizations identify possible patterns.

At the same time, we won't go into this analysis blindly, but have identified a preliminary set of questions that we want to study. In ecosystems, as an example, how topics are present in parts of the ecosystem and not in other parts might matter. Some topics may be isolated to a single system (or even to a part of this system), while others may be spread over several parts of the ecosystem. This might be indicative of aspects related to the modularization of the system, and might highlight key differences between topics that are well isolated, versus topics that cut across projects in the ecosystem.

We are also interested in looking at instances in which some of the topics change drastically; as these might highlight interesting aspects of the evolution. For example, topics that change when an underlying technology or framework is replaced or updated might be representative of what some experts have labeled as "implementation level topics" [14]. Furthermore, we are interested in looking at which topics change as a result of possible large scale refactoring efforts. Our lens of topic evolution could even serve as tool to assess the effectiveness of a refactoring effort. For instance, we could quantify if the changes before, versus after the refactoring, have a higher match with the topics that were affected by the refactoring (i.e., if the refactoring was successful, we would expect changes that occur afterward to match more closely to the topics that were refactored).

4.3 Developers and Topics

In the last study, we take a different look at the evolution of topics by additionally considering how developers are related to the topics in terms of the code they have changed. While previous research has looked at how developers are related to each other in terms of the topics they have changed [11], our perspective in this study is much broader. We are interested in finding patterns regarding the topics that a developer works on, considering changes that span multiple projects in the ecosystems.

We again have identified a set of possible questions, this time focusing on how developers work with topics in an ecosystem. For instance, research shows that developers often contribute work to several of the projects that are part of an ecosystem [9]. Our intuition dictates that it would be beneficial for a developer to gain expertise working with code related to certain topics, and to reuse this expertise throughout the ecosystem. We might find evidence of this phenomenon if we observe that developers recurrently make changes to code related to the same topics throughout the ecosystem.

We are also interested in exploring how a developer's expertise influences the topics they work on. Open source communities are usually described as having a hierarchical structure made up of different types of contributors according to their level of participation [9]. Core developers, for instance, are those that have been active for the longest and that contribute most of the changes to a project. We are interested in finding out if core developers tend to focus their work on certain topics, while other types of developers, for instance active developers (borrowing terminology from the onion model) work on other topics. The topics that core developers work on might be related to concerns that developers would consider to be more critical to the system, or perhaps more complex to modify and maintain. Conversely, the topics on which active developers work on might be related to concerns that are simpler, of less relevance to core of the system, or more appropriate for less experienced developers.

As a final example of our initial directions in this study, we are interested in observing how social features that support the interaction between developers might impact the evolution of topics. For instance, software ecosystem hosting sites such as GitHub go beyond project hosting to provide a social network of developers and their projects. In this network, previous research has identified the presence of highly influential developers, or "rock star" developers [4]. These developers are highly followed in the community, and any changes they make to the code are observed and analyzed by the community. We are interested in finding out how the topics in a system change when a highly influential developer starts contributing code. Moreover, will other developers adopt these topics, and to what extent? By answering these questions, we will be able to provide insight as to how expertise and knowledge is shared in these complex environments.

5. CONCLUSIONS

This work aims to explore the role of topic location techniques in understanding the evolution of a software ecosystem. To do so we will perform a series of studies using a novel fine-grained topic evolution model to observe patterns in the evolution of topics, and explore how developers have worked with these topics.

The patterns we identify will yield new insight into the design of tools that leverage topic location. Our results, for instance, could provide lessons for the development of tools that recommend related artifacts according to document to topic relationships. Most current tools that use topic modeling assume all topics to be equal. A novel recommendation tool could behave differently if a developer is modifying code related to a modularized topic versus one that cuts across the ecosystem. For modularized concerns, other parts of the code related to the same topic within the scope of a single project would perhaps be most useful. But for crosscutting topics it may be useful to recommend parts of the code, related to the same topic, from several other projects that are also part of the software ecosystem.

Moreover, the identification of interesting phenomena regarding how developers work with topics can also inform the design of better tools. For instance, recent work regarding development practices in GitHub [4] refers to new ways in which developers leverage social information to find projects of interest and identify useful technical knowledge that they want to bring to their project. The results of our research can support the development of tools that search and recommend related projects and code to a developer according to the topics similar to those they usually work on. Expanding on this idea, the tool could also help in onboarding by incrementally recommending projects and tasks that slowly drift from what the developers knows best, thereby continuously challenging them and broadening their skills and familiarity with the code base.

6. ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under grant number IIS-1111446.

7. REFERENCES

[1] Balagopalan, A. 2012. *Improving topic reproducibility in topic models*. University of California Irvine.

[2] Biggers, L.R. et al. 2012. Configuring latent dirichlet allocation based feature location. *Empirical Software Engineering*. 17, (Aug. 2012), 1–36.

[3] Blei, D.M. et al. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*. 3, (2003), 993–1022.

[4] Dabbish, L. et al. 2013. Leveraging Transparency. *IEEE Software*. 30, 1 (Jan. 2013), 37–43.

[5] Gonzalez-Barahona, J.M. et al. 2009. Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering*. 14, 3 (Nov. 2009), 262–285.

[6] Griffiths, T.L. and Steyvers, M. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*. 101, Suppl 1 (Apr. 2004), 5228–5235.

[7] Iansiti, M. and Levien, R. 2004. The keystone advantage: what the new dynamics of business ecosystems mean for strategy, innovation, and sustainability. *Harvard Business Review Press.* (2004), 255.

[8] Jansen, S. and Cusumano, M. 2012. Defining software ecosystems: a survey of software platforms and business network governance. *International Workshop on Software Ecosystems* (Cambridge, MA, 2012), 40–58.

[9] Jergensen, C. et al. 2011. The Onion Patch : Migration in Open Source Ecosystems. *19th symposium and 13th European conference on Foundations of software engineering* (Szeged, Hungary, 2011), 70–80.

[10] Linstead, E. et al. 2008. An application of latent dirichlet allocation to analyzing software evolution. *7th International Conference on Machine Learning and Applications* (San Diego, CA, 2008), 813–818.

[11] Linstead, E. et al. 2007. Mining eclipse developer contributions via author-topic models. *4th International Workshop on Mining Software Repositories* (Minneapolis, MN, May. 2007), 30–30.

[12] Lopez, N. and Hoek, A. Van Der 2013. Do topics characterize development tasks? *Submitted to WCRE 2013* (2013).

[13] Lukins, S.K. et al. 2010. Bug localization using latent dirichlet allocation. *Information and Software Technology*. 52, 9 (Sep. 2010), 972–990.

[14] Maskeri, G. et al. 2008. Mining business topics in source code using latent dirichlet allocation. *1st India Software Engineering Conference* (Hyderabad, India, 2008), 113–121.

[15] Mei, Q. 2005. Discovering evolutionary theme patterns from text - an exploration of temporal text mining. *11th Conference on Knowledge Discovery in Data Mining* (2005), 198–207.

[16] Mockus, A. 2009. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. *6th International Conference on Mining Software Repositories* (Vancouver, Canada, May. 2009), 11–20.

[17] Nistor, E.C. and Van der Hoek, A. 2009. Explicit concern-driven development with ArchEvol. *International Conference on Automated Software Engineering* (Aukland, New Zealand, 2009), 185–196.

[18] Thomas, S.W. et al. 2011. Modeling the evolution of topics in source code histories. *8th working conference on Mining software repositories* (Honolulu, HI, 2011), 173.

[19] Wettel, R. and Lanza, M. 2008. Visual Exploration of Large-Scale System Evolution. *15th Working Conference on Reverse Engineering* (2008), 219 – 228.