

---

# Software Analysis with Unsupervised Topic Models

---

Erik Linstead<sup>1,2</sup>, Lindsey Hughes<sup>2</sup>, Cristina Lopes<sup>1</sup>, Pierre Baldi<sup>1</sup>

<sup>1</sup> School of Information and Computer Sciences. University of California, Irvine.

<sup>2</sup> Department of Math and Computer Science. Chapman University, Orange, CA.  
elinstea@ics.uci.edu, hughe120@mail.chapman.edu, lopes@ics.uci.edu, pfbaldi@ics.uci.edu

## Abstract

We provide an overview of our work in applying unsupervised topic and author-topic models based on Latent Dirichlet Allocation (LDA) to the problem of mining large software repositories at multiple levels of granularity. Our approaches allow us to automatically discover the topics embedded in code and extract document-topic and author-topic distributions. In addition to serving as a convenient summary for program content and developer activities, these and other related distributions provide a statistical and information-theoretic basis for quantifying and analyzing software complexity, developer similarity, and the evolution of software over the release timeline.

## 1 Introduction

Large repositories of software, available publicly over the Internet or privately within large organizations, create several new challenges and opportunities for statistical data mining and machine learning. One fundamental challenge is to develop approaches for automatically measuring, monitoring, and better understanding software content, complexity and temporal evolution, as well as to gain insight into the relationship between humans and the software they produce. Here we address this challenge by adapting statistical topic modeling methods, in particular Latent Dirichlet Allocation [1], to automatically extract topics from source code. These topics provide an intuitive and probabilistic basis for program understanding. Furthermore, we leverage document-over-topic and topic-over-document distributions to derive a precise, operational, definition of scattering and tangling, two fundamental metrics for software complexity proposed in the Aspect-Oriented Programming (AOP) [2] paradigm. We employ the same distributions to model the temporal evolution of software, effectively applying topic models to identify refactoring events on the software release timeline. Finally, we apply probabilistic Author-Topic (AT) models [3] to the task of identifying developer contributions from source code.

While our methods are very general, here we apply them to a sample of projects written solely in Java. Specifically, we download 12,151 Java projects from Sourceforge and Apache and filter out distributions packaged without source code (binaries only). The end result is a repository consisting of 4,632 projects, containing 366,287 source files, with 38.7 million lines of code, written by 9,250 developers. Additionally, to study the evolution of software releases we employ the codebases of several versions of Eclipse. Our results demonstrate the effectiveness and flexibility of unsupervised topic models in the software engineering domain, and make progress toward the long-term goal of informing and guiding software development, engineering, and management.

## 2 Methods

A widely held hypothesis of the software engineering community is that software consists of concerns, or aspects, that represent the various functional concepts represented in source code, and that

Table 1: Representative scattering results for the full repository.

Concern	Extracted Topic from LDA	Entropy
Exception Handling	'exception illegal argument runtime pointer'	0.796
Logging	'error log debug string throwable'	0.790
Persistence	'byte read write bytes short'	0.747
Plotting	'category range domain axis paint'	0.641
Security	'field string security underlying leg'	0.496
Scientific Computing	'long address gsl short matrix'	0.466

these concerns can transcend package, class, and source file boundaries. Concerns (functional concepts) that are manifested in many source files are said to be “scattered”. Similarly, source files that contain many concerns are said to be “tangled”. While scattering and tangling are largely accepted as major indicators of software complexity, mathematically-grounded, unsupervised techniques for extracting concerns and quantifying scattering and tangling have so far remained elusive.

There is a strong conceptual similarity between latent topics and the concepts of concerns in software engineering. So much so, that we propose to unify the concept of latent topic with the concept of concern in the domain of software, with individual source files acting as documents and the identifier tokens within them defining the vocabulary. Thus software concerns can be identified by running the LDA algorithm properly adapted to software data, an adaptation that consists of processing source code based on software vocabulary heuristics (splitting based on camelcase, etc.) to produce the document-word assignments at the heart of the algorithm [4]. Once the topics of a software project are identified by the LDA approach, a formal definition and quantification of scattering and tangling can be derived from the corresponding topics-over-files and files-over-topics distributions and their entropies. For example, if the distribution of topic  $t$  across modules  $m_0 \dots m_n$  is given by  $p^t = (p_0^t \dots p_n^t)$  then the scattering of topic  $t$  can be measured by the entropy  $H(p^t)$  of this distribution. Likewise, the tangling of a file is measured by the entropy of its distribution over topics. These methods can naturally be extended to monitor the temporal evolution of software.

### 3 Results

In this section we present the results obtained by applying topic modeling techniques to identifying and measuring scattering and tangling, as well as studying software temporal evolution. While space constraints allow for the inclusion of only a small representative subset of results, the reader may refer to our previous work in this area for detailed descriptions of our methods and their complete results and validation for extracting concerns [5], validating the Aspect-Oriented hypothesis [6], and analyzing the dynamics of software.

Table 1 shows a small sample of extracted topics and entropy scattering results obtained with the topic modeling approach applied to the full repository described above. Normalized entropy takes a value between 0 and 1, and represents the uncertainty associated with the random variable representing a given topic’s distribution over source files. Intuitively, topics with high entropy are more pervasive than those with lower entropy, with an entropy of 1.0 representing a topic with a uniform distribution over all files and an entropy of 0 representing a topic assigned to only one file. Looking at the topics one can see that various software concepts are represented, ranging from general subjects like exception handling to specific domains such as graph plotting.

The first three high-entropy examples in Table 1 correspond precisely to the concerns that have been used as prototypical examples for AOP such as exception handling, logging, and persistence. The fact that their entropy is high across the entire repository tells us that these topics are pervasive across software projects. In other words, concerns such as string manipulation, exception handling, and so on, recur quite frequently in the 366,287 source files of the repository. The other examples in Table 1 are less representative of general cross-cutting concerns and more domain-specific. We undertake a detailed study of domain-specific concerns in [6].

Table 2 provides a very small sample of tangling results obtained with LDA topic modeling applied to the full repository, highlighting five files with high entropy and five files with low entropy. Three of the five files with high entropy correspond to report generation. In looking at these files, we

Table 2: Representative tangling results for the full repository.

File	Entropy
org/openharmonise/rm/commands/CmdGenerateReport.java	0.826
it/businesslogic/ireport/gui/ReportQueryDialog.java	0.789
mail/core/org/columba/mail/imap/IMAPServer.java	0.788
jRivetFramework/webBoltOns/ReportWriter.java	0.787
org/lnicholls/galleon/apps/musicOrganizer/MusicOrganizer.java	0.766
doctorj-5.0.0/org/incava/java/ASTNestedClassDeclaration.java	0.338
net/sf/farrago/namespace/jdbc/MedJdbcColumnSet.java	0.228
com/planet_link/coffee_mud/Exits/Door.java	0.000
buoy/event/FocusLostEvent.java	0.000

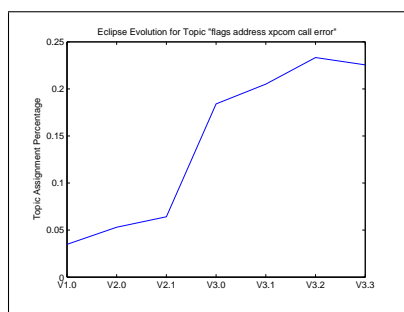


Figure 1: Emergence of XPCOM functionality over 7 Eclipse releases.

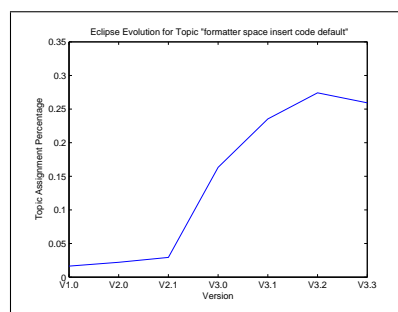


Figure 2: Emergence of formatting functionality over 7 Eclipse releases.

noticed that they include string manipulation, database access, exception handling, interface design, and a wide variety of other concepts. The same can be said for the remaining high entropy files, which constitute an IMAP server (.788) and a music organizer (.766), both of which are noticeably complicated in implementation. At the other end of the spectrum, the low entropy files correspond to very specific functional modules, such as handling nested class declaration in an abstract syntax tree (.338) and column manipulation database code (.228). The table also gives examples of two files with a tangling of 0. Door.java originates from the CoffeeMud game engine, and is assigned only to the topic ‘mob environmental can stats room.’ Similarly, FocusLostEvent.java, a specific event listener for capturing mouse focus, is assigned only to the topic ‘event mouse component focus cursor.’ Visual inspection of these files confirms their functional simplicity.

Equally as important as managing software complexity is the ability to summarize, in an unsupervised fashion, points on the release timeline where major software functionalities are integrated. Our tools can be applied to this task as well, and here we demonstrate it using seven successive releases of Eclipse. To this end, we run LDA on all seven releases at once, producing a document-topic matrix,  $M$ , where  $M(i, j)$  corresponds to the number of times document  $i$  was assigned to topic  $j$ . For each topic we aggregate the number of assignments for each software release. Dividing by the total number of assignments yields a percentage representing the extent to which a topic is manifested in each release. By plotting these percentages on the release timeline and examining the resulting graphs for spikes, one can identify the appearance or disappearance of concerns across versions. For example, Figure 1 shows the evolution in Eclipse for the topic ‘flags address xpcom call error,’ which is related to the XPCOM protocol, a remote method invocation technology similar to CORBA. The spike from version 2.1 to version 3.0 indicates that most of the functionality related to XPCOM was added in version 3.0, which is supported by inspection of the source code. Figure 2 shows a similar result for a topic related to code formatting, which also appears to have been largely integrated in version 3.0.

Turning to the identification of developer contributions, a representative subset of 4 author-topic assignments extracted via AT modeling on 2,119 source files from Eclipse 3.0 is given in Table 3. To the right of each topic is a list of the most likely authors for each topic with their probabilities. Examining the topic column of the table it is clear that various functions of the Eclipse framework are

represented. For example, topic 1 clearly corresponds to unit testing, topic 2 to abstract syntax trees, topic 3 to building projects, and topic 4 to automated code completion. Remaining topics range from package browsing to compiler options. Examining the author assignments for the various topics provides a simple means by which to discover developer contributions and infer their competencies. It should come as no surprise that for Eclipse the most probable developer assigned to the JUnit framework topic is “egamma”, or Erich Gamma. In this case, there is a 97% chance that any source file in our dataset assigned to this topic will have him as a contributor. Based on this rather high probability, we can also infer that he is likely to have extensive knowledge of this topic. This is a particularly attractive example because Erich Gamma is widely known within the software engineering community for being a founder of the JUnit project, a fact which lends credibility to the ability of the topic modeling algorithm to assign developers to reasonable topics. One can interpret

Table 3: Representative Topics and Authors from Eclipse 3.0.

#	Topic	Author Probabilities	#	Topic	Author Probabilities
1	junit	egamma 0.97065	3	nls-1	darins 0.99572
	run	wmelhem 0.01057		ant	dmegert 0.00044
	listener	darin 0.00373		manager	nick 0.00044
	item	krbarnes 0.00144		listener	kkolosow 0.00036
	suite	kkolosow 0.00129		classpath	maeschli 0.00031
2	ast	maeschli 0.99161	4	token	daudel 0.99014
	button	mkeller 0.00097		completion	teicher 0.00308
	cplist	othomann 0.00055		current	jlanneluc 0.00155
	entries	tmaeder 0.00055		identifier	twatson 0.00084
	astnode	teicher 0.00046		assist	dmegert 0.00046

the remaining author-topic assignments along similar lines. For example, developer “daudel” is assigned to the topic corresponding to automatic code completion with probability .99. Referring back to the Eclipse bug data it is clear that the overwhelming majority of bug fixes for the codeassist framework were made by this developer. Such information is useful in determining assignments for future bug fixes. Moreover, the author-topic distributions may be used directly to perform developer similarity analysis using KL divergence, as demonstrated in [4].

In this paper we have provided an overview of several analysis problems in the software engineering community that have benefited from the application of unsupervised topic models such as LDA. These models provide a scalable, statistical basis for program comprehension, software complexity and evolution, and identifying programmer contribution. While space limitations allow us to provide only a highlight of progress we have made in this area, the reader is encouraged to refer to our other work for detailed results, further applications, and a comprehensive literature review of related research in this area.

## References

- [1] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, January 2003.
- [2] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.
- [3] M. Steyvers, P. Smyth, M. Rosen-Zvi, and T. Griffiths. Probabilistic author-topic models for information discovery. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, New York, NY, USA, 2004. ACM Press.
- [4] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Min. Knowl. Discov.*, 18(2):300–336, 2009.
- [5] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. Mining internet-scale software repositories. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 929–936, Cambridge, MA, 2008. MIT Press.
- [6] Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead, and Sushil K. Bajracharya. A theory of aspects as latent topics. In *OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 543–562, New York, NY, USA, 2008. ACM.