Tracking the Flow of Ideas through the Programming Languages Literature

Michael Greenberg*[1], Kathleen Fisher[2], and David Walker[3]

1 **Pomona College**
2 **Tufts University**
3 **Princeton University**

Abstract

How have conferences like ICFP, OOPSLA, PLDI, and POPL evolved over the last 20 years? Did generalizing the Call for Papers for OOPSLA in 2007 or changing the name of the umbrella conference to SPLASH in 2010 have any effect on the kinds of papers published there? How do POPL and PLDI papers compare, topic-wise? Is there related work that I am missing? Have the ideas in O'Hearn's classic paper on separation logic shifted the kinds of papers that appear in POPL? Does a proposed program committee cover the range of submissions expected for the conference? If we had better tools for analyzing the programming language literature, we might be able to answer these questions and others like them in a data-driven way. In this paper, we explore how *topic modeling*, a branch of machine learning, might help the programming language community better understand our literature.

1998 ACM Subject Classification I.2.7 Natural Language Processing — Text analysis

Keywords and phrases programming languages literature, topic models, irony

1 Introduction

Over the past half-century, the programming language community has developed a foundational new science, the science of software, that now governs much of our world. One of the legacies of this great work is a vast literature — a literature so large no one person can know more than the tiniest fraction of it. Unfortunately, our normal interaction with this literature is tragically limited. At best, we read a small selection of papers in our area, look at their related work sections and follow the citations back a level or two. Occasionally, a friend will alert us to a neat paper that she has read. That is about it.

If we had better tools for analyzing the programming language literature, there are so many more interesting questions we could ask and answer:

- How have the topics at PL conferences evolved over the last 20 years?
- Did generalizing the OOPSLA call for papers broaden the conference?
- How has O'Hearn's classic paper on separation logic influenced POPL?
- Where did the themes touched by my paper appear in the past?

In addition to answering questions like these for the sake of curiosity, tools better able to analyze the programming language literature might be of help to the chairs of programming language conferences and the editors of programming language journals. Program chairs would like to know that their PCs cover the appropriate topics effectively. Editors would like reviewers with sufficient expertise, but, perhaps, different perspectives.

* This work was done while Michael was still a postdoc at Princeton University.

SNAPL.
Editor: Shriram Krishnamurthi; pp. 1–16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Many of these questions are deep semantic questions about the true flow of ideas through the programming language literature and beyond. We cannot answer these questions with perfect accuracy or certainty, but we may be able to approximate them. To investigate this possibility, we have begun to explore an analysis technique developed in the machine learning community, known as *topic models* [4]. Generally speaking, topic models are probabilistic descriptions of the words that appear in a document or a collection of documents. Such models provide insights into the topics that pervade a corpus and the relative weights of those topics. We can use these models to compare individual papers or groups of papers, and we can track the ebb and flow of topics over time. Such models have recently been used to analyze articles from the journal *Science*, the *Yale Law Review*, and other archives of documents [4]. They have also been used to explore the links betweeen academic literature and funding [15].

In this paper, we describe our initial efforts to apply these models and techniques to the programming language literature and to ask questions of interest to the programming language community. For data, we assembled a corpus of 4,355 abstracts from four of the top programming languages conferences (ICFP, OOPSLA, PLDI, and POPL) and the full text of all 2,257 POPL and PLDI papers that have appeared as of February 2015. We have generated models of this corpus as a whole and for the conferences individually and compared them. We have also analyzed individual documents and compared papers based on their models. We have built a web-based tool (available from `tmpl.weaselhat.com`) that allows other researchers to browse the programming language literature by topic, find related papers, and visualize the evoluation of the topics studied in PL over time. Source code for our tools is available on github.[1] Although our results to date are intriguing, this paper marks only the beginning of the project; many interesting questions remain.

The remainder of this paper is organized as follows. We first describe our methods (Section 2). Specifically, we give a tutorial introduction to *Latent Dirichlet Allocation* topic models and we describe how we curated our corpus. We next present our preliminary results (Section 3) and discuss future work (Section 4). We end with a discussion of related work (Section 5) and concluding thoughts (Section 6).

2 Methodology

2.1 Topic Models

To explain the ideas behind the simplest topic model, the Latent Dirichlet Allocation (LDA), we adapt the explanation given by David Blei [4] to a programming-languages setting. In this model, each document is considered to be a mixture of some number of topics. For example, consider the paper "Gradual Typing for First-Class Classses" [17] shown in Figure 1, which describes a type system for gradually converting dynamically-typed object-oriented programs into statically typed programs in a companion language. We manually highlighted various words occuring in the abstract: yellow for words related to components, green for words related type systems, and blue for words related to Object-Oriented Programming (OOP). If we highlighted the entire paper in this way, it would be apparent that the document mixes concepts from these three topics: components, type systems, and OOP.

LDA models this intuition that a document is a mixture of different topics. In the LDA framework, each "topic" is represented as a probability distribution over a fixed collection of

[1] See `https://github.com/mgree/sigplan/`.

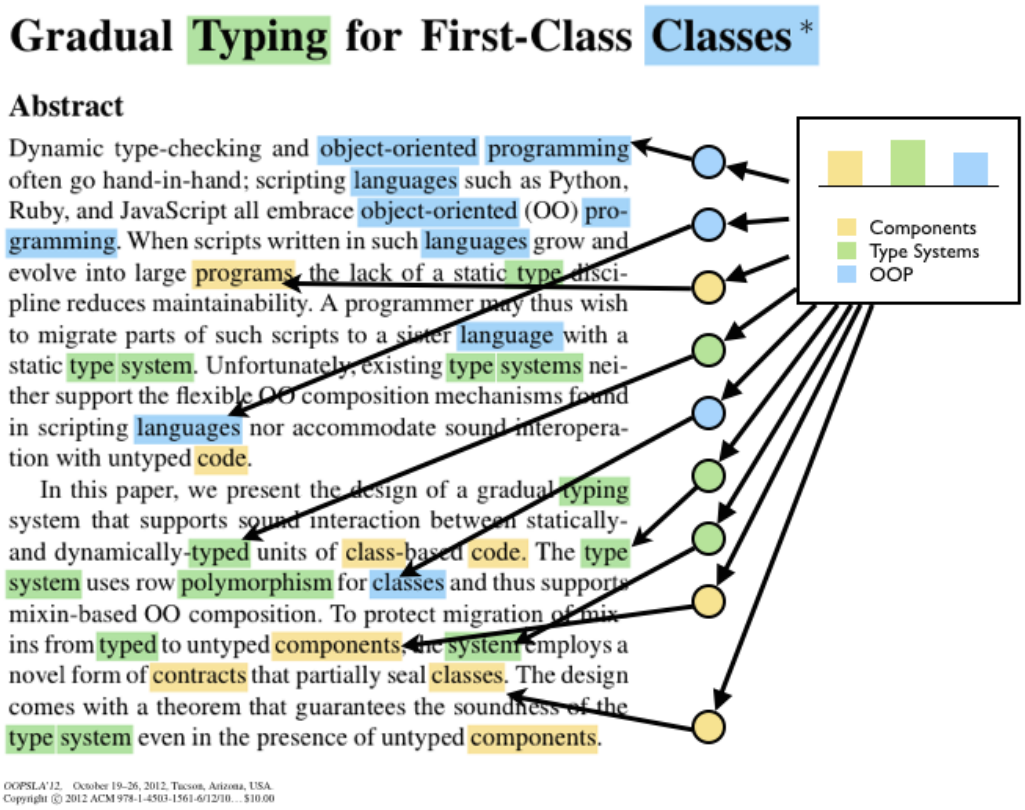


Figure 1 LDA topic modeling is based on the generative process illustrated here. It works over a fixed collection of topics assumed to be shared by all documents in the corpus (top right). First, we select a probabilty distribution over the topics (histogram). Then, to produce each word in the document, we first choose a topic from the distribution (the colored circles) and then we choose the actual word from the word distribution associated with the topic (the highlighted words).

words. The probability associated with each word indicates how strongly the word belongs to the topic. For example, words like 'type' and 'polymorphism' have high probabilities in the "Type System" topic, while 'component' and 'contract' have high probabilities in the "Components" topic. In contrast, words like 'cpu' and 'register' have low probabilities in both topics. A document is, in turn, modelled as a distribution over such topics.

In the paper in Figure 1, the probability distribution over topics would assign weights to the topics "Components," "Type Systems," and "OOP." The chart at the right of the figure shows this distribution, with "Type Systems" carrying slightly greater weight than either "Components" or "OOP." The words in the paper are drawn from the probability distributions associated with these topics. To put it another way, if we wanted to generate this document, or one similar to it, we would pick each word for the document independently according to the following *generative process*: (1) pick a topic (according to the topic distribution) and then (2) pick a word from that topic (according to its distribution over words). Other papers in the SIGPLAN corpus would exhibit a different mix of topics. A key point is that all the documents in an LDA framework share the same collection of topics, just mixed in different proportions (including nearly zero).

Of course the purpose of topic modeling is not to generate random documents but rather to automatically discover the topics present in a corpus. We *observe* the documents, but the

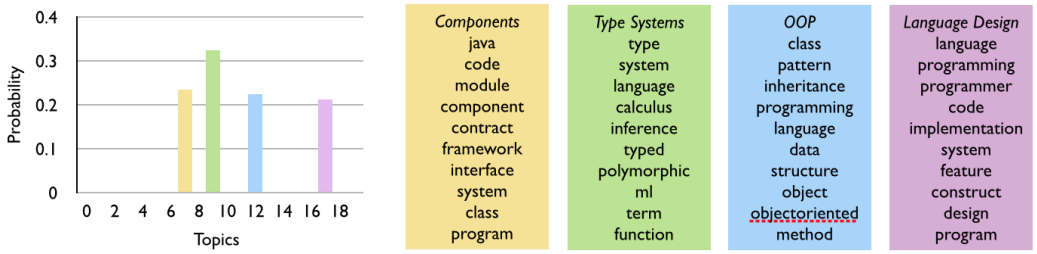


Figure 2 LDA inference for 20 topics run on a corpus of 4,355 titles and abstracts from SIGPLAN conferences and applied to the example paper "Gradual Typing." The probability distribution at the left shows the inferred topic mixture for the paper. The boxes at the right show the 10 most probable words for the four topics present in the paper.

topic structure is *hidden*, including the set of topics, the topic distribution for each document, and the per-document, per-word topic assignment. The challenge of this area of research is to reconstruct the hidden topic structure from an observed corpus, essentially running the generative process backward, attempting to answer the question: What fully-defined topic model most likely explains the given corpus?

We ran David Blei's LDA-C[2] over a corpus of 4,355 titles and abstracts from ICFP, OOPSLA, PLDI, and POPL papers that are available in the ACM Digital Library, having set the number of topics at 20.[3] This process produced a topic model M, judged to be the most likely 20-topic model to explain the SIGPLAN corpus given a set of 20 "seed" papers, randomly selected by LDA. We then applied M to the "Gradual Typing" paper to impute the topic distribution that best explains the content of that paper. The results appear in Figure 2. This histogram on the left shows that although M could have assigned any probability distribution to the twenty topics, it selected only four. The right side of the figure shows the most likely words associated with those four topics. Looking at these words, we can see that three of the topics correspond to the ones we higlighted in Figure 1: Components, Type Systems, and OOP. LDA identified a fourth topic, whose words suggest it might be called "Language Design." Note that LDA does not produce the titles for the topics; we manually came up with names for them based on the most likely words (see Section 2.4). Automatically inferring such titles is an open research problem.

2.2 The corpora

We used the SIGPLAN research papers stored in the ACM Digital Library (DL)[4] to build two distinct corpora for this project. Our *abstract corpus* contains one "document" for every paper from every available year of ICFP (1996-2014), OOPSLA (1986-89,1991-2014), PLDI (1988-2014), and POPL (1973,1975-2015). Each such document is comprised of the concantenation of the paper title, its author list, and its abstract. The scraping process we used to build this corpus is imperfect — it doesn't find every paper listed in the DL (see Section 2.6). Our *full-text corpus* contains one document for each of the 2,257 POPL and

[2] `http://www.cs.princeton.edu/~blei/lda-c/`
[3] One of the limitations of LDA is that it will not help decide how many topics appear in a corpus. The user must decide in advance and seed the inference mechanism with the chosen number.
[4] `http://dl.acm.org/`; SIGPLAN conferences are available from `http://dl.acm.org/sig.cfm?id=SP946`

PLDI papers that have appeared as of February 2015. We used `pdftotext` from the Xpdf tool suite[5] to convert the `pdf` files from the ACM DL to raw text. For each document in both corpora, we separately store metadata, recording its author list along with when and where it was published.

Once we had compiled the raw data for the two corpora, we converted it into a form that LDA-C can read: a document file that represents each document as a "bag of words." Parsing forces us to decide which words are interesting and how to count those words. For example, the word 'we' appears many times in academic writing, but it isn't a meaningful topic word in the way that 'type' or 'flow' or 'register' are. So that we may focus on meaningful words, we drop all words on a *stopword* list from the document. We started with a standard stopword list for English,[6] amending it with two lists of words. First, we added words which appeared in more than 25% of the documents in our corpus. We also added words that felt irrelevant, such as 'easily', 'sophisticated', 'interesting', and 'via'.

Once we winnowed out the stopwords, we had to turn each text into a bag of words: a list pairing words with the number of times they occur. This process raises more questions requiring judgment: Are 'type' and 'types' different words? What about 'typing' and 'typical'? To answer such questions, we counted words as the same when they reduced to the same uninflected word via NLTK's[7] WordNet [19] lemmatizer. In this case, 'type' and 'types' both reduce to the uninflected word 'type', but 'typing' and 'typical' are left alone. Finally, we coalesced hyphenated words into single words, e.g., 'object-oriented' became 'objectoriented' and 'call-by-value' became 'callbyvalue'.

2.3 Our topic models

Topic models are parameterized by a number k, which represents the desired number of topics; at the moment, no machine learning techniques exist to automatically determine the appropriate number of topics. To explore this space, we used LDA-C to build topic models for both corpora for k equal to 20, 50, 100, and 200.

On the abstract corpus, a typical run of LDA-C with 20 topics takes about 2 minutes on a four-core 2.8Ghz Intel i7 with 16GB of RAM and a solid-state hard drive; simultaneous runs with 200, 300, and 400 topics take just shy of two hours. On the full-text corpus, generating the topic model for k=50 requires under an hour. It takes roughly 8 hours to generate the topic model for k=200.

Once LDA-C has processed our corpus, it produces files indicating how much each word belongs to a topic and how much each document belongs to each topic. We translate LDA-C's output to three files:

– a human-readable listing of the top 10 words and top 10 papers for each topic;
– a spreadsheet collating the metadata for each document (title, authors, year, and conference) with that document's topic weights; and
– a spreadsheet aggregating the total topic weights for each year of each conference.

The human-readable listing of top words and papers for each topic was a vital check on the sanity of LDA-C's output as we debugged our process; we also used it to assign names to the topics when we made graphs.

[5] `http://www.foolabs.com/xpdf/download.html`
[6] `https://pypi.python.org/pypi/stop-words/2014.5.26`
[7] `http://www.nltk.org/`

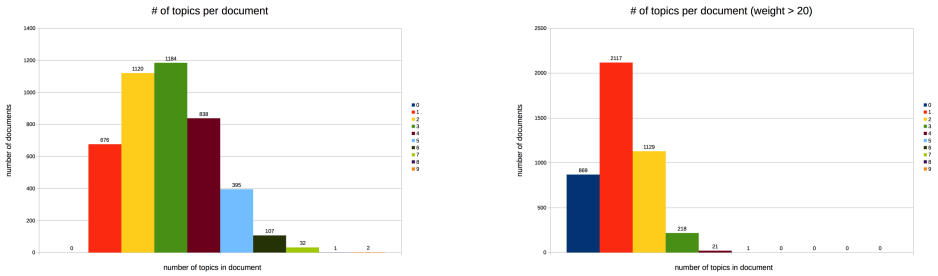


Figure 3 Number of topics per paper (left: normalized per topic, right: normalized per topic, minimum topic weight of 20)

Register Allocation	Program Transformation	Concurrency	Parsing
register (-3.6)	program (-4.0)	thread (-3.5)	grammar (-3.9)
instruction (-4.4)	loop (-4.5)	lock (-4.4)	attribute (-4.0)
code (-4.5)	variable (-4.6)	operation (-4.5)	string (-4.6)
variable (-4.7)	statement (-4.8)	memory (-4.6)	tree (-4.6)
graph (-4.9)	assertion (-4.9)	read (-4.6)	set (-4.6)
figure (-4.9)	value (-5.0)	execution (-4.7)	symbol (-4.7)
value (-5.0)	expression (-5.1)	program (-4.9)	language (-4.7)
loop (-5.0)	procedure (-5.2)	access (-5.0)	production (-4.7)
node (-5.0)	true (-5.3)	data (-5.0)	pattern (-4.8)
range (-5.0)	condition (-5.3)	concurrent (-5.0)	parser (-4.8)

Figure 4 Consensus topic names and top ten keywords (with log-likelihood for that word in that topic) for selected topics from the k=20 topic model on the full-text PLDI-POPL corpus.

2.4 Validating the topic models

Our topic models are sane: every paper is assigned *some* topic weights (Figure 3). On the left, we see that every document has non-minimal (0.03189) weight in *some* topic; on the right, we see that nearly one in five papers (869 out of 4,355) doesn't have a weight higher than 20 in any topic. Running with more topics reduces the number of indeterminate, topic-less documents. However, more topics can also lead to over-fitting and difficulty in distinguishing the underlying concepts. For comparison, Blei's analysis of the journal Science used 100 topics; Analysis of the Yale Law Review used 20 topics. We have experimented with up to 400 topics, but elected to present 20 for simplicity here.

Topic models do not provide names for the inferred topics, just lists of key words and the strength of the association between each paper and the topic. As another validity check, each of us independently named each topic for k=20 by looking at the keywords and highest ranking papers. Each of the lists were very similar; for most topics, it was a straightforward process to produce a consensus name for each topic. Figure 4 shows these consensus names and the associated top ten keywords for the k=20 topic model of the PLDI-POPL full-text corpus. We found topic 12 (given the title Program Transformation) from this list puzzling. At first glance, the keywords seem like a plausible if generic grouping, but the papers seem unrelated. A closer inspection of the top-ranked papers reveals that they all involve program transformations, particularly transformations related to numeric programs. This topic seems to be an example of a topic type that we had hypothesized but not previously seen: one where the papers are grouped by approach rather than the problem they are trying to solve.

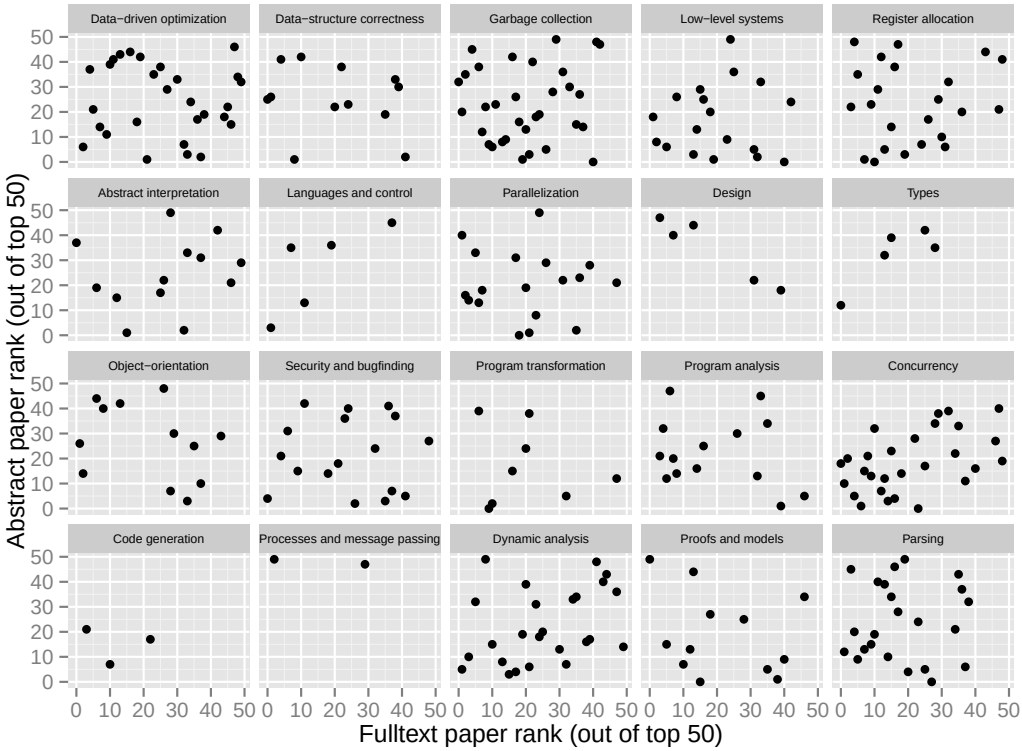


Figure 5 Common papers in the top 50 papers of LDA models built using abstracts and full text, by topic

2.5 Abstract vs full-text models

What documents should we feed to LDA? We have two corpora at our disposal: the abstracts from ICFP, OOPSLA, PLDI and POPL (with some omissions — see Section 2.6); and the full text of PLDI and POPL. Once a topic model has been built, it can be used to classify documents of any size. So how should we build our models?

We ran LDA with k=20 on two corpora drawn from PLDI and POPL: one used abstracts, one used full text. In general, both models produced similar paper rankings in each topic (Figure 5). Each dot in Figure 5 represents a paper that appears in the top 50 papers in both models for a given topic. The x-axis is the paper's rank in the full-text model; the y-axis is the paper's rank in the abstract model. Prefect agreement would be a diagonal line of dots. Most topics show a fair to high agree of general agreement, though some show very little. Hearteningly, every topic shows *some* agreement.

We haven't yet answered the methodological question — perhaps existing research in machine learning can guide us. We use both models to explore the data in this paper.

2.6 Limitations and impediments

One limitation of our current work is that our corpora include only data from four programming language conferences. There are many other high-quality venues that publish relevant work, including ASPLOS, CC, CONCUR, ECOOP, ESOP, SAS, *etc.* We have omitted them only because adding more data sources is labor intensive. We felt that the data from the

four conferences we have chosen was sufficient for an initial exploration.

Problems with data quality begin at the ACM. The ACM DL is missing data. The abstracts for ICFP, OOPSLA, PLDI, or POPL are missing for 1991. POPL is missing abstracts from 1996 through 1999. The ACM DL listings don't obviously differentiate hour-long invited keynote talks and the shorter talks in the sessions. We chose to include abstract-less keynotes as "documents"... but is that the right decision?

Scraping the DL for abstracts is not completely straightforward. Some papers are clearly in the HTML but our scraper doesn't detect them. Ideally, we would have access to the database underlying the DL, with programmatic access to metadata and the full text of each paper. Even then, extracting full text from PDFs isn't foolproof: math can show up as garbage characters that confuse LDA, or whole sections can fail to appear.

3 Results

While topic models can be used to analyze the programming languages literature in several different ways, we have focused primarily on two broad questions. The first question concerns how the topics present in the various programming languages conferences have evolved over time. The second involves the use of topic models to search for related work.

3.1 Characterizing the conferences

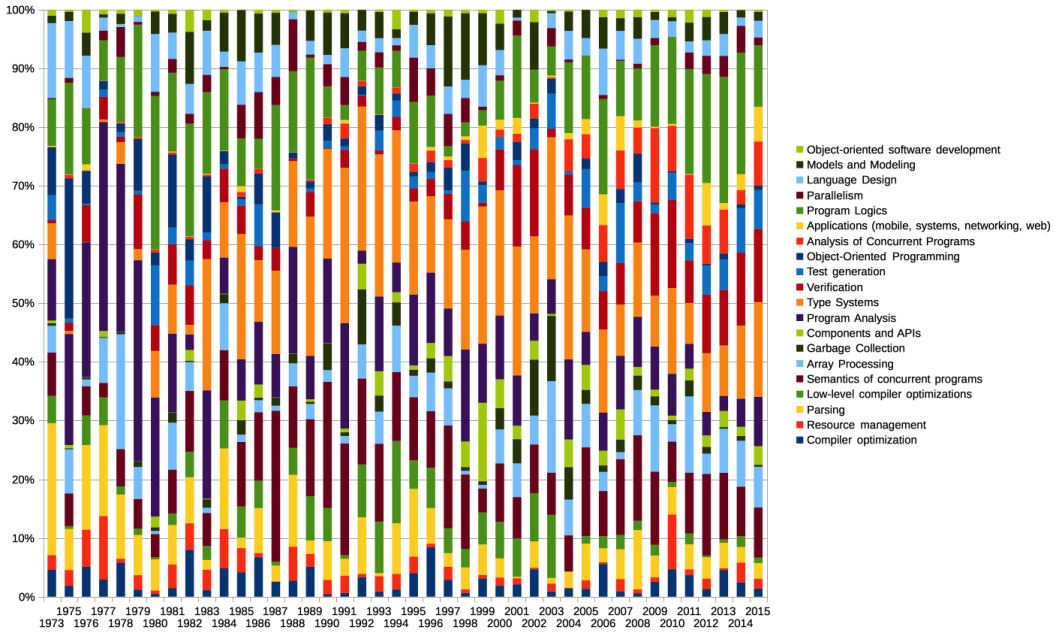
The prevailing wisdom is that each of the four major SIGPLAN conferences has its own character. Topic models highlight some of the distinctions between conferences as well as some of the similarities. They also provide insight into some of the major changes in research focus that have occurred over the last 40 years.

The four conferences, holistically.

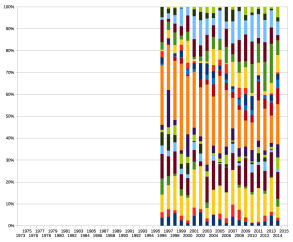
In our first experiment, we separated papers into collections according to conference (ICFP, OOPSLA, PLDI and POPL) and year (1973-2015). In each conference, in each year, we can apply a topic model to determine the topics and weight at which they appear in that year.

Figure 6 presents the results for our 20-topic model generated from the abstracts drawn from the 4 conferences. The top chart represents research presented at POPL between 1973 and 2015. Each column represents a year; the length of a colored line represents the weight of the corresponding topic in the given year. The three smaller charts present data for ICFP, OOPSLA and PLDI respectively. When viewing these charts online, we recommend zooming in to inspect the details.

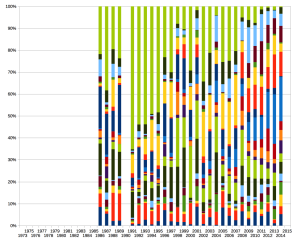
Analysis of these charts confirms many broad intuitions about the topics that dominate certain conferences — the longer the line of a particular color, the more content from the corresponding topic appears in the conference. For instance, ICFP and POPL are both home to many papers on type systems, the bright orange topic that runs through the middle of the diagrams. OOPSLA and PLDI contain less content on these topics. PLDI is best known for its focus on classic compiler optimizations (the dark blue at the bottom of each chart) or low-level compiler optimization (the dark green 4th from the bottom). None of the other conferences contain such an emphasis on those topics. And as we will see in more detail later, there has been a dramatic reduction in the number of publications in the area, even at PLDI. OOPSLA, unsurprisingly, is best known for its papers on object-oriented software development, the light green dominating the top of the OOPSLA chart (and displayed in minute quantities at the other conferences). Interestingly, even at OOPSLA, we see a



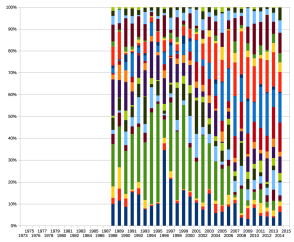
(POPL)



(ICFP)



(OOPSLA)



(PLDI)

Figure 6 Holistic, year-by-year topic model plots for four programming languages conferences. Topic model uses abstracts.

remarkable decrease in the prevalence of this topic beginning in the early 2000s through to 2015.

The four conferences, by topic.

The previous charts present the big picture at each major conference and provide a means to compare the weights of different topics against one another. However, we also found it interesting to isolate the topics and to compare the conferences to each other on a topic-by-topic basis. Figure 7 presents a series of 20 graphs, one for each topic that we learned from the abstracts of the four conferences. The title we chose for each topic is presented at the top of each graph. The x-axis represents the year (ranging from 1973-2015) and the y-axis represents the weight of the topic. Each line represents a different conference: ICFP (red), OOPSLA (green), PLDI (turquoise) and POPL (pink). The grey shadow surrounding each line provides an estimate of the error involved in drawing the curve (the tighter the band,

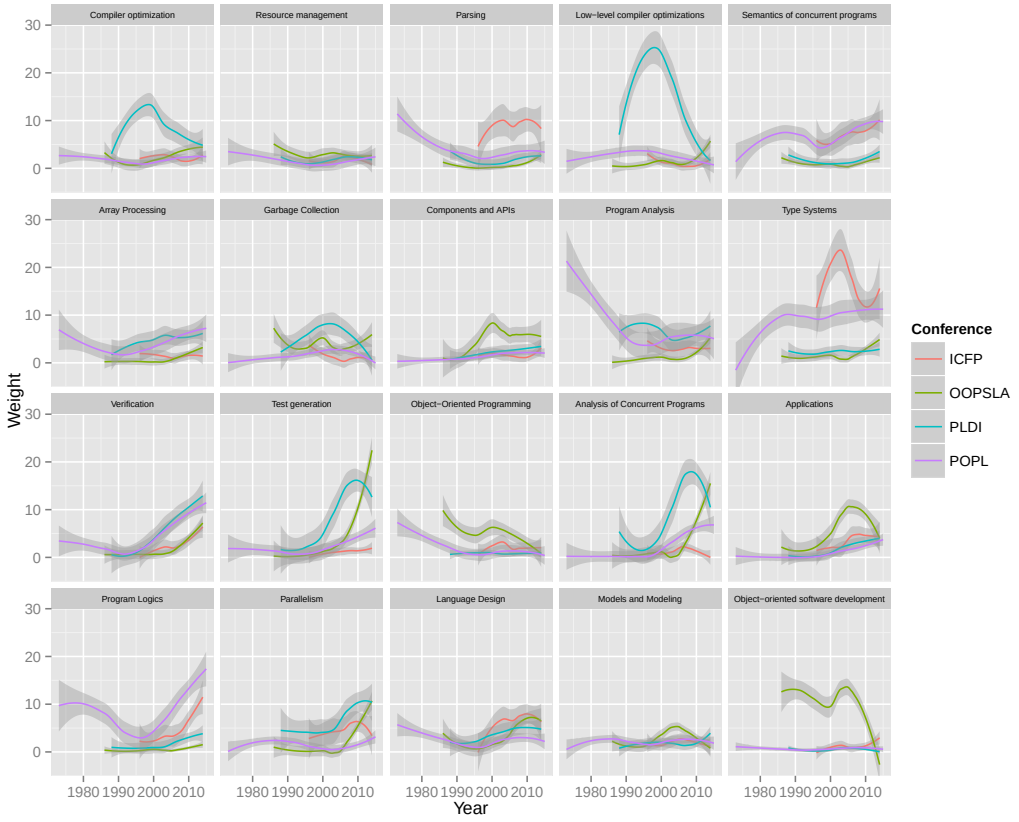


Figure 7 Topic-by-topic plots comparing four programming languages conferences. Topic model uses k=20 and abstracts.

the better the estimate).[8]

Before beginning the experiment, one question we had was whether the change in the Call For Papers for OOPSLA in 2007 and the subsequent rebranding in 2010 of the umbrella event as SPLASH correlated with any changes in OOPSLA content. One of the goals of the Call for Papers changes was to broaden OOPSLA and deemphasive object-oriented programming. To consider this question, two topics, object-oriented programming (3rd row, 3rd column in Figure 7) and object-oriented software development (4th row, 5th column) are particularly relevant. Object-oriented programming pertains to analysis of object-oriented programming language features and the use of those features. It is characterized by keywords (in order of relevance) class (significantly more relevant than the others), pattern, inheritance, programming, language, data, structure, object, object-oriented, and method. Object-oriented software development pertains to software engineering methods and program design. It is characterized by the words design, software, object-oriented, development, programming, object, system, process, tool and project. The two most relevant documents are both panels: *The OO software development process*, a panel from OOPSLA 1992 [6] and

[8] Each curve is generated using Loess smoothing over the set of points generated from that topic for each paper in the conference in question. We used the smoothing package from R's graphical library (stat_smooth(method=loess)). See `http://rgm3.lab.nig.ac.jp/RGM/R_rdfile?f=ggplot2/man/stat_smooth.Rd&d=R_CC` for details.

Agile management - an oxymoron?: Who needs managers anyway?, a panel from OOPSLA 2003 [1]. Both panels have abstracts that were included in the OOPSLA proceedings in their respective years. When it comes to object-oriented programming, the charts show there was a steady decline in content at OOPSLA beginning in around the year 2000, when it was at a local maximum at the end of the dot-com boom. This decline persisted through the changes to OOPSLA's rebranding with little apparent dip. When it comes to object-oriented software development, there was a peak in the mid-2000s near the time of the rebranding, and a rather preciptious drop-off thereafter. One other topic that shows a similar trend is the topic on applications, which includes mobile, systems, web and network programming.

Since investigation of elements of object-oriented programming and software development have been on the decline at OOPSLA over the past decade or so, a natural question to ask is what topics have been taking their place? The answer includes topics such as test generation, analysis of concurrent programming, parallelism, and verification. Each of those topics show a significant spike at OOPSLA since the mid-2000s. It may well be that OOPSLA's rebranding encouraged more submissions along these lines.

Another trend that emerges from these graphs involves topics surrounding program optimizaton. The compiler optimization topic (row 1, column 1) and the low-level compiler optimization topic (row 1, column 4) both show spikes through the 90s and strong downward trend through the 2000s. OOPSLA has seen a slight uptick in low-level compiler optimization since roughly 2010, but across all four programming languages conferences, research in optimization appears down significantly.

PLDI vs POPL, by topic.

PLDI and POPL are broadly perceived as the two most prestigious conferences in programming languages. At any given time, the topics they support heavily are bound to be at the center of programming languages research.

Figure 8 presents a series of charts similar to those we just studied in Figure 7. However, here we focus strictly on POPL and PLDI, and we generated a model using the full text of each paper in each conference.

To begin, there are many interesting similarities between the two conferences. For instance, on topics such as abstract interpretation, design, object orientation, program transformation, program analysis, code generation, and parsing, PLDI and POPL have followed very similar trends. Some of those trends are unsurprising: POPL had a substantial presence in parsing in the 70s, which has continuously decreased. PLDI followed suit shortly after its inauguration. Both PLDI and POPL had a larger presence in code generation that has dropped off. On the other hand, algorithmic and symbolic techniques for static program analysis (titled abstract interpretation) have steadily increased in both conferences since the mid-nineties. Here, one might speculate that the advent of practical SAT and SMT tools may have helped push this topic forward.

On the other hand, there are some substantial differences between the conferences. When it comes to types, PLDI has been relatively stable over the course of its existence. POPL's weight in this topic has steadily increased over the years. Another interesting trend involves language definition and control structures (row 2, column 2), which involves analysis of topics such as continuations, partial evaluation and lazy execution. At POPL, there was a huge spike during the 90s, which has since waned. At PLDI, these topics never really made much of an appearance. POPL also sees a lot of activity in highly theoretical topics involving proofs and semantic models for programs. On the other hand, at PLDI, topics such as parallelization and dynamic analysis are persistently higher than at POPL.

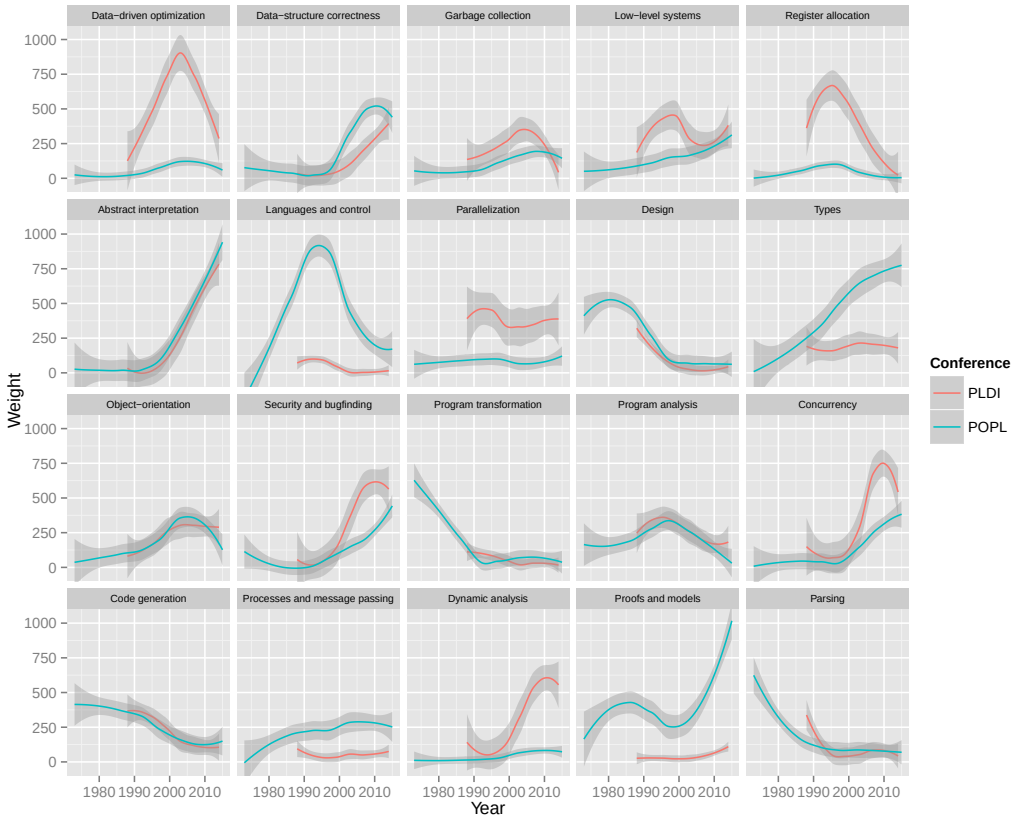


Figure 8 Topic-by-topic plots comparing PLDI and POPL. Topic model uses k=20 and full text.

3.2 Finding your friends

Topic models offer a similarity measure, allowing us to compare documents or collections of documents to one another. When two documents (or sets of documents) are assigned topic vectors that are close in a geometric sense, they contain similar topic distributions. To study the similarity measure generated by the topic models, we compared work cited by a paper to a random selection of POPL and PLDI papers. Our hypothesis was that models for related work would usually be closer to the model for a given paper than randomly selected papers would be.

To test our hypothesis, we selected four papers (one involving each of the authors, and one more) — *Concurrent Data Representation Synthesis* by Hawkins et al. [10] (CDRS), *Proof-Carrying Code* by George Necula [12] (PCC); *From System F to Typed Assembly Language* by Morrisett et al. [11] (TAL); and *Space-Efficient Manifest Contracts* by Greenberg [9] (SEMC). For each paper, we inferred a topic vector using LDA with k=20 and a corpus built on abstracts. We then inferred topic vectors for each paper's citations.[9] Figure 9 displays the results of our analysis. More specifically, for each paper (CDRS, PCC, SEMC,

[9] We excluded one citation from Greenberg — his thesis. Since it's *much* longer than a conventional conference paper, the thesis is 'far' from the paper only because all of its topic weights are larger in an absolute sense, even though the topic vector is largely in a similar direction. From the other papers, we also excluded several cited books, theses or other documents that were either vastly longer than a conference paper or unobtainable.

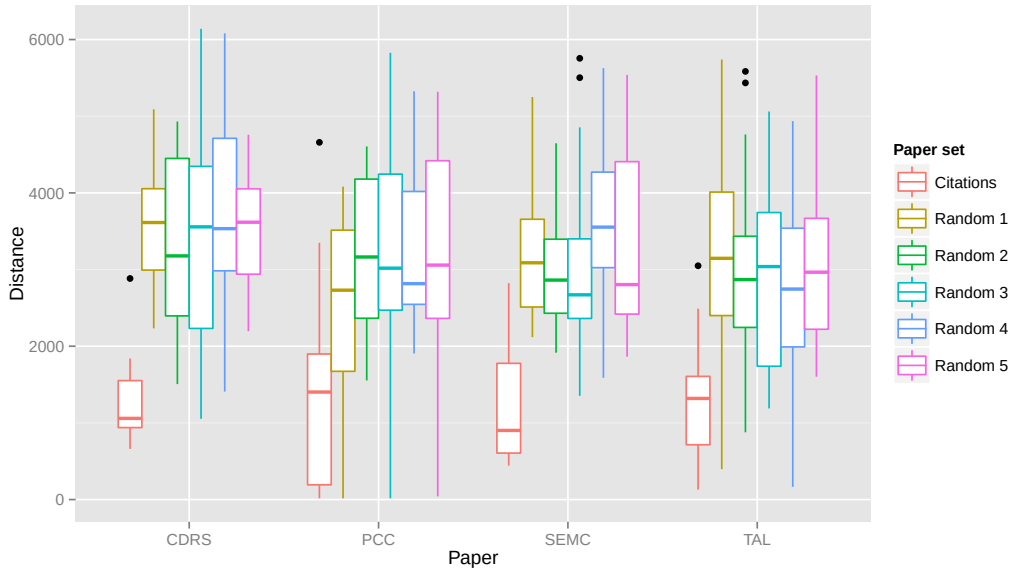


Figure 9 Are citations closer in topic space than random papers? Topic model uses abstracts.

TAL), the left-most boxplot (in red) shows the distribution of Euclidean distance between each paper and its citations. In such boxplots, the third quartile, the top of the boxplot contains 75% of cited papers below it, while the bottom of the boxplot demarcates the 25% boundary. The height of the box is the interquartile range (IQR) and the vertical lines stretch 1.5*IQR from the top or the bottom of the box. Dots beyond those lines are outliers. The papers CDRS, SEMC, TAL have low medians and third quartiles, indicating that the bulk of their related is relatively close to the paper. In all these cases, 75% of cited papers are closer to the original paper than 75% of the random papers.

We included the PCC paper to illustrate that not all papers in our data set have particularly useful models. The PCC paper is one such paper. According to our model, it is relatively "topic-less" — its topic vector is quite short and close to the origin — so it's hard to use its topic vector to classify related work. Indeed, its related work overlaps a little more with the randomly generated papers.

Having established that papers are relatively close to their related work in topic space, we can flip this idea on its head: given a paper, we can find papers in our corpus that are close to it — these may be related work! We've implemented a prototype tool for finding related work at `http://tmpl.weaselhat.com`. You can upload a PDF and we will identify the twenty most related papers from PLDI and POPL.

At present, our related work tool is simple: it runs a single model on a fixed corpus. With more data, we could offer more potential suggestions. With more computational power and engineering, we could use a suite of models (with different document seeds, number of topics, or data sources) to "boost" our related work search.

Even in this most primitive form, our tool has produced useful and surprising results. On his first test of the system, David Walker put in some of his collaborators' recent work — Osera and Zdancewic's *Type-and-Example-Directed Program Synthesis* [14] — and found a related paper by Nix entitled *Editing by example* (POPL 1984) that Osera and Zdancewic were unaware of [13]. Older papers like this one may be difficult to find via conventional

search.[10] Michael Greenberg put in a recent networking workshop paper [8] and found some interesting work that solves a related problem in a different area [7]. We find these early anecdotal results encouraging. We hope that topic modeling can provide help with automated related work search driven by *documents* rather than *keywords*.

4 Future Work

One goal of this work is to build a website offering programming language researchers access to the tools and/or analysis results we find are most interesting or effective. Our prototype is up at `http://tmpl.weaselat.com/`. The source is freely available at `https://github.com/mgree/sigplan/`, allowing other communities to reuse our work.

Understanding the conferences.

There are a number of ways we can paint a more detailed picture of our programming languages conferences. First, we can simply try running LDA with more topics. Second, we can investigate richer topic models. For instance, dynamic LDA [3] models the shift of topic content over time, and has been used for *lead/lag* analysis: When new topics appear, do they appear in one conference and spread to others? One very interesting study looked at the relationship between the topics funded and the reearch that appeared at conferences [15]. We could obtain the summaries of funded NSF proposals (and possibly other institutions) to determine if we can identify relationships between funded topics and conference content.

Understanding the researchers.

We can also use topic models to learn topics for researchers, not just documents. There are many ways to do this — from collating all of a researcher's oeuvre to using time-aware techniques, like dynamic LDA [3]. Given a topic model for each researcher, it is possible to construct several additional tools. For instance, to help ensure a program committee has interests distributed similarly to the expected distribution of papers for a conference, we could check that the aggregated topic weights of the PC's publications are similar to the aggregated topic weights of the papers from the past year's submissions. To chose which papers to review, PC or ERC members can find papers with topics similar to their own — an idea that has already been explored in the Toronto Paper Matching System [5].

We may also be able to use models of PL researchers to answer questions about how researchers change over time. Do they stay in one topic, or work in many topics? Are there researchers who lead their conferences? Can we apply social network models to co-authorship graphs to understand how people share interest in topics? Can this help in assembling workshops or Dagstuhl-like seminars?

5 Related work

There is much work in the machine learning community aimed at understanding large collections of documents. David Blei's web page [2] on topic modeling contains pointers to several survey papers, talks and software packages. We were also inspired by the work of

[10] Google keyword searches including "program synthesis from examples" and "synthesizing string transformations from examples" do not display this result in the first three pages of links returned. More recent research floats to the top.

Charlin *et* al. [5], the architects of the Toronto Paper Matching System, which aims to match reviewers to papers using topic models. This prior work is all quite general; we aim to use topic-modeling software to study the flow of ideas across the programming languages literature specifically, something which has not be done before using similar methods.

There are various other ways one might analyze the programming languages literature. For instance, in a recent article for CACM [16], Singh *et. al.* studied research in hardware architecture between 1997 and 2011 on the basis of occurrences of keywords. Measuring keyword occurrences has the benefit of simplicity, but it does not identify collections of words — themes, as Blei would call them — that find themselves colocated in documents. These themes may help us characterize papers and researchers succinctly and may suggest interesting measures to compare them. Moreover, the LDA algorithm identifies these themes for us without anyone having to specify them.

TMVE [18] is a topic model visualization engine that offers a baseline expectation of how users might interact with a topic model. It produces a static website for a single topic model, while we envision more dynamic interactions on our website, like our prototype of related-work search.

6 Conclusion

Topic models offer the potential to give us a new lens on the programming languages literature. We already have some interesting results analyzing the content of our major conferences through this lens and we believe there are many more interesting questions to attempt to answer with these techniques.

Acknowledgments

This work is supported in part by the NSF under grant CNS-1111520. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1 Lougie Anderson, Glen B. Alleman, Kent Beck, Joe Blotner, Ward Cunningham, Mary Poppendieck, and Rebecca Wirfs-Brock. Agile management - an oxymoron?: Who needs managers anyway? In *OOPSLA*, pages 275–277, 2003.

2 David Blei. Topic modelling. `http://www.cs.princeton.edu/~blei/topicmodeling.html`, 2015.

3 David Blei and John Lafferty. Dynamic topic models. In *International Conference on Machine Learning*, 2006.

4 David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

5 Laurent Charlin and Richard Zemel. The Toronto Paper Matching System: An automated paper-reviewer assignment system. In *ICML Workshop on Peer Reviewing and Publishing Models*, June 2013. See also `http://papermatching.cs.toronto.edu/webapp/profileBrowser/about_us/`.

6 Dennis de Champeaux, Robert Balzer, Dave Bulman, Kathleen Culver-Lozo, Ivar Jacobson, and Stephen J. Mellor. The OO software development process (panel). In *OOPSLA*, pages 484–489, 1992.

7 Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, Sriram Rajamani, and Damien Zufferey. P: Safe asynchronous event-driven programming. In *Programming Language Design and Implementation (PLDI)*, 2013.

8 Marco Gaboardi, Michael Greenberg, and David Walker. Type systems for SDN controllers, 2015. PLVNET.

9 Michael Greenberg. Space-efficient manifest contracts. In *Principles of Programming Languages (POPL)*, 2015.

10 Peter Hawkins, Alex Aiken, Kathleen Fisher, Martin Rinard, and Mooly Sagiv. Concurrent data representation synthesis. In *Programming Language Design and Implementation (PLDI)*, 2012.

11 Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. In *Principles of Programming Languages (POPL)*, 1998.

12 George C. Necula. Proof-carrying code. In *Principles of Programming Languages (POPL)*, 1997.

13 Robert Nix. Editing by example. In *Principles of Programming Languages (POPL)*, 1984.

14 Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In *Programming Language Design and Implementation (PLDI)*, 2015.

15 Xiaolin Shi, Ramesh Nallapati, Jure Leskovec, Dan McFarland, and Dan Jurafsky. Who leads whom: Topical lead-lag analysis across corpora. In *NIPS Workshop*, 2010.

16 Virender Singh, Alicia Perdigones, and Fernando R. Mazarrón José Luis Garcia, Ignacio Cañas-Guerroro. Analyzing worldwide research in hardware architecture, 1997-2011. *CACM*, 58(1):76–85, 2015.

17 Asumu Takikawa, T. Stephen Strickland, Christos Dimoulas, Sam Tobin-Hochstadt, and Matthias Felleisen. Gradual typing for first-class classes. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pages 793–810, New York, NY, USA, 2012. ACM.

18 TMVE: Topic model visualization engine. `https://code.google.com/p/tmve/`, 2015.

19 Princeton University. Wordnet. `http://wordnet.princeton.edu`, 2010.