Trend Analysis on the Metadata of Program Comprehension Papers

Matúš Sulír and Jaroslav Porubän

***Abstract*—As program comprehension is a vast research area, it is necessary to get an overview of its rising and falling trends. We performed an n-gram frequency analysis on titles, abstracts and keywords of 1885 articles about program comprehension from the years 2000–2014. According to this analysis, the most rising trends are feature location and open source systems, the most falling ones are program slicing and legacy systems.**

***Index Terms*—Program comprehension, bibliography, trends, n-grams.**

I. Introduction

PROGRAM comprehension deals with an understanding of existing programs by developers. It is a vast research area, ranging from studying mental models to the design and evaluation of reverse engineering tools. For this reason, it is necessary to gain an overview of the field – its methods and techniques. It is important to become familiar not only with stable knowledge, but also with the latest trends.

We have the following research questions:

- **RQ1:** What are the most rapidly rising trends in program comprehension?
- **RQ2:** What are the falling trends in program comprehension?

II. Method

A brief overview of the process is depicted in Fig. 1.

A. Data Collection

First, we searched the citation databases Scopus and IEEE Xplore for the following exact terms (in quotes):

- program comprehension,
- program understanding,
- code comprehension.

Only articles published between the years 2000 and 2014 (inclusive) were searched. The Scopus queries were continuously refined to exclude unrelated entries, e.g. articles about TV program comprehension or a chemical paper where the words "program" and "understanding" were only accidentally subsequent. In Scopus, excluding artifacts like proceedings cover pages, tables of contents and author indexes was easily accomplished by limiting the document types to journal articles and conference papers. For IEEE Xplore, an extensive query based on string matching had to be constructed.

Neither Scopus nor IEEE Xplore contained metadata from the last year of IEEE International Conference on Program Comprehension (ICPC 2014) which could negatively affect the validity of our study. They were available only in the ACM Digital Library, which does not offer an option to mass-download metadata and even prohibits it. Fortunately, HCI Bibliography, hosted by ACM SIGCHI, offered them[1].

We downloaded 1599 entries from Scopus in the BibTeX format, 740 from IEEE Xplore as a CSV (comma-separated values) file and 43 from the HCI Bibliography website in the EndNote format. We did not utilize IEEE Xplore's BibTeX export as it is limited to 100 entries and does not distinguish between author-supplied and automatically assigned keywords. All exported files were converted if needed and combined into one BibTeX file. Ten incomplete items (without authors or an abstract) and 487 duplicates were removed. This gives us a total of 1885 analyzed articles.

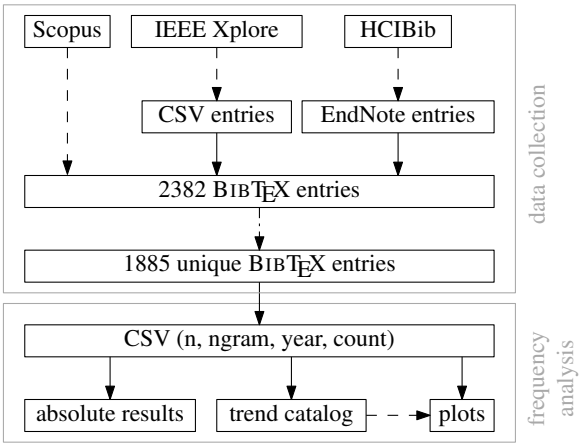

Fig. 1. A block diagram of the analysis method. Solid arrows represent automated processes, dash-dotted semiautomatic and dashed manual ones.

B. Conversion to an N-gram List

Each article in a bibliographic database is called an *entry*. Every entry has *fields* like an article title or year published.

For the purpose of this study, we analyzed titles, abstracts and author-supplied keywords using a technique called n-grams. An n-gram with a length n is a sequence of n words in a text. For example, the sentence "Here you are" contains

Manuscript received March 30, 2015; revised May 12, 2015. This work was supported by VEGA Grant No. 1/0341/13 Principles and methods of automated abstraction of computer languages and software development based on the semantic enrichment caused by communication.

Matúš Sulír (matus.sulir@tuke.sk) and Jaroslav Porubän (jaroslav.poruban@tuke.sk) are with the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice.

[1] http://hcibib.org/ICPC14

 This is the accepted version of: M. Sulír, J. Porubän. Trend analysis on the metadata of program comprehension papers. Engineering of Modern Electric Systems (EMES) 2015, pp. 153–156. http://doi.org/10.1109/EMES.2015.7158425

unigrams “here”, “you” and “are”; *bigrams* “here you” and “you are” and a *trigram* “here you are”.

First, each abstract was split into a list of sentences to prevent n-gram analysis across sentence boundaries like recognizing the bigram “program comprehension” in the fragment “to program. Comprehension of”. Although titles usually contain only one sentence, they were split, too. In our study, every keyword was considered a separate sentence.

Next, the words “a”, “an” and “the” were removed from all sentences because the meaning of an n-gram like “the program” is the same as “a program”. A list of n-grams was produced for each sentence. We worked with n-grams with a length from 1 to 4 as longer n-grams are rarely repeated in texts.

From the list of n-grams, we removed ones which comprised of at least 50% stopwords. A *stopword* is a frequent function word, i.e. a word which does not convey any meaning on its own. For example, the n-gram “any of programs” was excluded while the phrase “comprehension of programs” was retained. We used a stopword list from the Snowball stemmer[2].

All n-grams with their associated origin information were transformed into a simple tabular model (a list of records) and saved into a CSV file. Each of its records contains these fields:

- n,
- n-gram,
- year,
- count.

For example, a record “2,dynamic analysis,2008,33” means that a bigram “dynamic analysis” occurred 33 times in a title, abstract or keywords of articles published in 2008.

Up to this point, we used the Ruby language (mainly because there is a convenient BibTeX library available[3]) to automatize some parts of the process.

C. Frequency Analysis

The produced CSV file was loaded by a script written in the statistical language R, which also generated the plots used in this article.

We defined a *frequency* of an n-gram of length n as follows:

$$freq(ngram, year) = \frac{count(ngram, length(ngram), year)}{count(*, length(ngram), year)}$$

where $count$ is a sum of the counts for records matching the given criteria and $*$ means “any n-gram”. For example, the frequency of a bigram “design pattern” for year 2001 is the count of occurrences of this n-gram in 2001 divided by the sum of occurrences of all bigrams in this year. This approach is similar to the one used by the current version of Google Ngram Viewer [4].

Using this metric, a PDF file which we call *Trend Catalog* was produced, containing year-frequency plots of about 800 most frequent n-grams in the whole database. We manually inspected the catalog to get an overview of the most interesting trends.

[2]http://snowball.tartarus.org/algorithms/english/stop.txt

[3]http://rubygems.org/gems/bibtex-ruby

[4]http://books.google.com/ngrams/info

It is often useful find out the frequency of multiple closely related n-grams, for example `slice+slices+slicing`. Thus we define the frequency of a list of n-grams as the sum of the frequencies of individual n-grams:

$$frequency(ngrams, year) = \sum_{x \in ngrams} freq(x, year)$$

Finally, it is desirable to put multiple competing n-grams (or n-gram lists) to one plot. Competing phrases are separated by commas. We can perceive this as a simple query language, similar to the one used by Google Ngram Viewer [1]. For example, a plot of the query `case study, experiment, review+survey` is shown in Fig. 2.

III. Absolute Results

Before exploring the actual trends, let us outline the results in terms of the absolute numbers of occurrences of the most frequent n-grams. The top three unigrams were “program”, “software” and “code”.

The most interesting were bigrams, shown in Table I. High positions of the phrases “reverse engineering” and “software maintenance” correspond with our previous statement that they are two most related research fields to program comprehension [2].

After inspecting the most used trigrams and 4-grams, we came to the conclusion that these phrases represent mainly clichés and bigrams complemented by insignificant words.

IV. Trends

A. Research Methods

We can see in Fig. 2 that case studies have a slightly decreasing tendency. There is a sudden rise of experiments in 2011, when they even outperformed case studies. Naturally, reviews and surveys are the least common types as they summarize existing research results.

TABLE I
The Most Frequent Bigrams

Order	Bigram	Count
1.	program comprehension	1541
2.	source code	1070
3.	program understanding	642
4.	reverse engineering	518
5.	software maintenance	357
6.	software systems	345
7.	software system	265
8.	software engineering	234
9.	dynamic analysis	208
10.	case study	191
11.	program slicing	180
12.	program analysis	170
13.	open source	168
14.	software development	164
15.	information retrieval	153

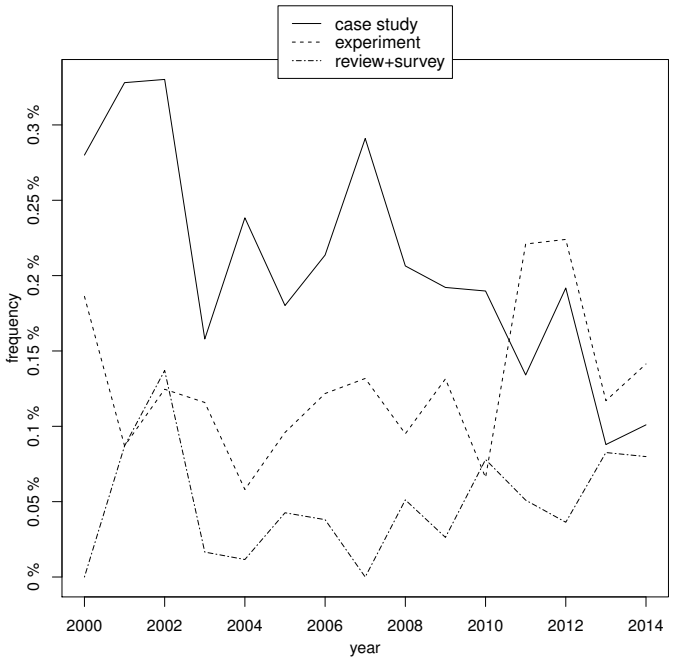

Fig. 2. Types of research in program comprehension.

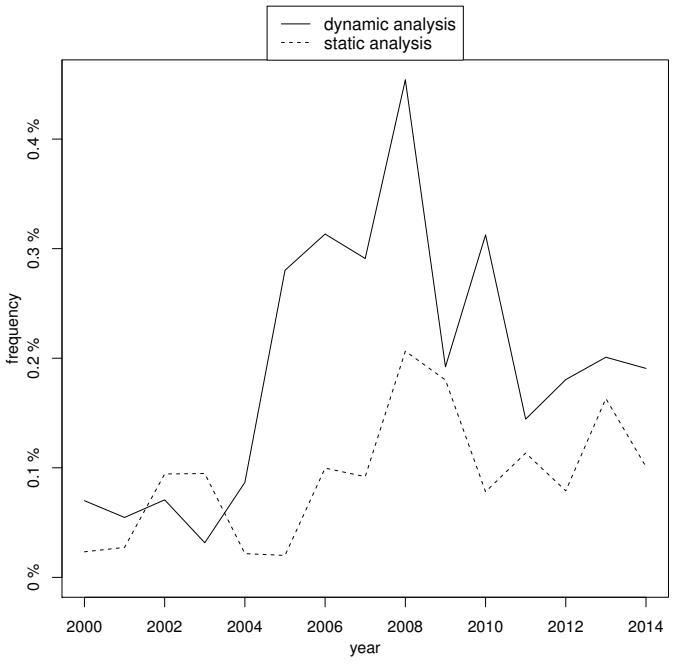

Fig. 3. Static vs. dynamic analysis.

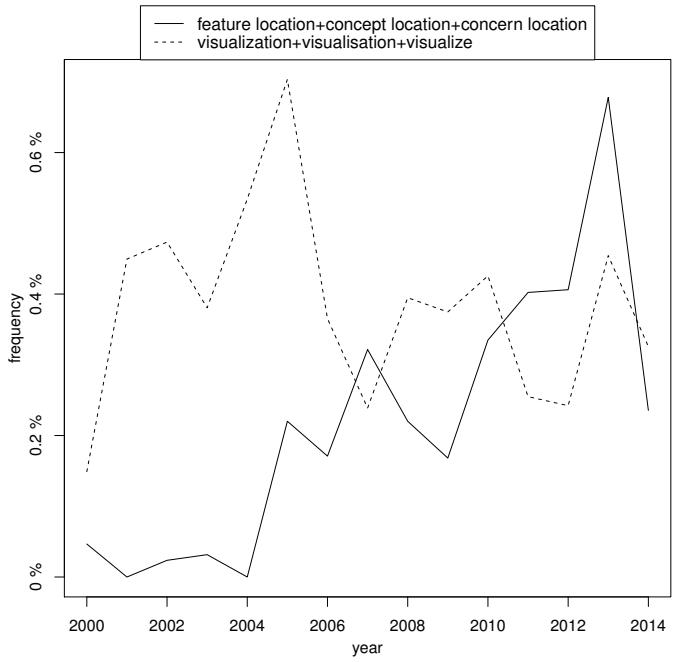

Fig. 4. Feature location vs. visualization.

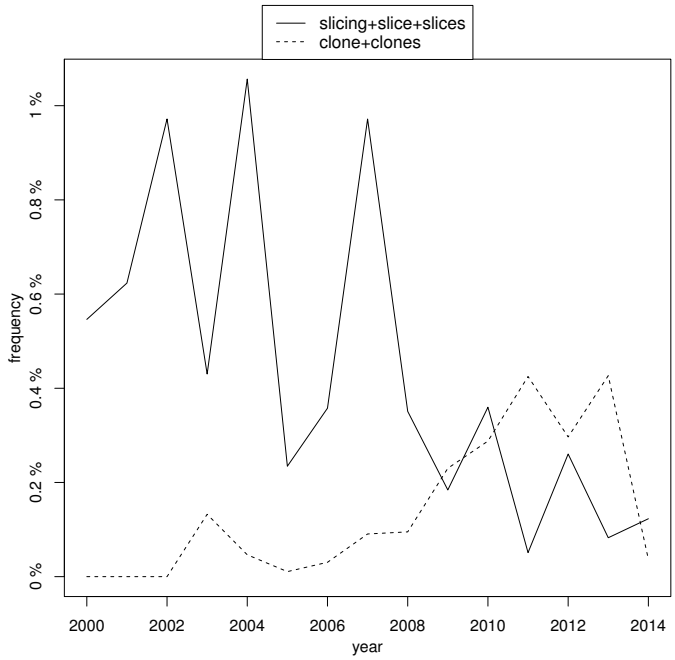

Fig. 5. Program slicing, code clone detection.

B. Techniques

Static program analysis (e.g., [3]) deals with the source code of an application without running it, whereas dynamic analysis (surveyed in [4] and exemplified in [5]) utilizes runtime information. As seen in Fig. 3, in 2004, the dynamic program analysis overtook the static one and this state lasts until today.

We can see a plot of two often used program comprehension techniques – feature location and visualization – in Fig. 4. While visualization is relatively steady, feature location rises rapidly from 2004.

Program slicing [6] is a technique used to find all code semantically related to the given statement or variable and produce a new program, containing only this related code. In Fig. 5, we can see a decreasing popularity of slicing.

Code clone detection [7] is a research field with a long tradition. However, from Fig. 5 it is obvious that it started to associate with program comprehension only in recent years.

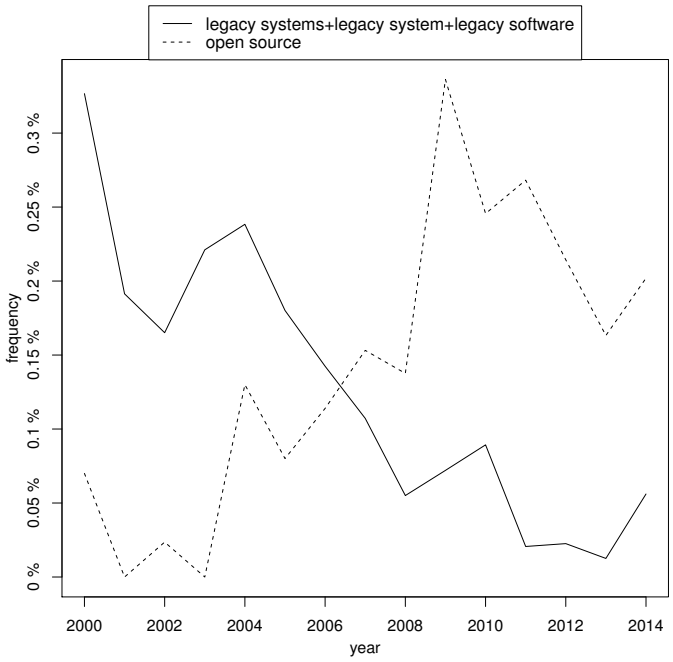

Fig. 6. Studied systems – legacy vs. open source.

C. Systems

In Fig. 6, we can see what types of systems the researchers study. Legacy systems have a clearly decreasing tendency during the last 15 years. One trend which immediately caught our attention is an extremely rapid increase of the n-gram "open source" between the years 2003 and 2009.

V. Threats to Validity

First of all, full texts of the articles were not analyzed. This is mainly due to the fact that many popular digital libraries forbid mass-downloading of the full texts. A notable exception, Scopus, has an option to download up to 50 articles at once. However, none of the few papers we tried to download were available in their full-text database. The most probable reason is the fact that publishers, which transferred the rights from the authors, often significantly limit publishing full-texts on other servers. A viable solution is open access publishing [8].

A relatively small number of articles per year to produce reliable results is another possible threat.

VI. Related Work

A similar trend analysis [9] was performed on the papers from nine Mining Software Repositories conferences. The authors analyzed another, albeit related, research field. Our study is methodologically inspired by this paper in some aspects, particularly the idea of n-gram frequency analysis. However, our approach differs in many ways. For example, the authors analyzed only nine years of one particular conference, while we performed a web search for the keywords. On the other hand, they analyzed full texts, while we only processed the metadata.

An n-gram query language syntax was inspired by Google Ngram Viewer [1]. We also considered directly using this tool for our study since it offers a text corpus many orders of magnitude larger than ours. While the time period is broader, too, it does not include the last two years. Furthermore, it is not specialized to computer science and the corpus consists of books, which are less suitable for latest trend analysis than research articles.

It is also possible to analyze Q&A (question and answer) sites such as Stack Overflow to find out trending topics. In [10], a technique called latent Ditrichlet allocation [11] was used to recover topics from natural language texts.

VII. Conclusion and Future Work

We analyzed the trending phrases in bibliographical references to see the rising and falling ones. To answer **RQ1**, the most rising trends are feature (and concept) location and the study of open source systems. Answering **RQ2**, program slicing and the study of legacy systems are the most falling trends.

Our next goal is to perform a manual systematic analysis [12] in selected subfields of program comprehension, using the acquired experience.

References

[1] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden, "Quantitative analysis of culture using millions of digitized books," *Science*, vol. 331, no. 6014, pp. 176–182, 2011.

[2] M. Sulír, "Program comprehension: A short literature review," in *SCYR 2015: 15th Scientific Conference of Young Researchers*, May 2015, to appear.

[3] J. Kollár, S. Chodarev, E. Pietriková, and Ľ. Wassermann, "Identification of patterns through Haskell programs analysis," in *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, Sept 2011, pp. 891–894.

[4] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 684–702, Sept 2009.

[5] M. Bačíková, J. Porubän, and D. Lakatoš, "Defining domain language of graphical user interfaces," in *2nd Symposium on Languages, Applications and Technologies*, ser. OpenAccess Series in Informatics (OASIcs), J. P. Leal, R. Rocha, and A. Simões, Eds., vol. 29. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 187–202.

[6] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, "A brief survey of program slicing," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 2, pp. 1–36, Mar. 2005.

[7] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470 – 495, 2009, special Issue on Program Comprehension (ICPC 2008).

[8] P. O. Brown *et al.*, "Bethesda statement on open access publishing," June 2003. [Online]. Available: http://www.earlham.edu/~peters/fos/bethesda.htm

[9] S. Demeyer, A. Murgia, K. Wyckmans, and A. Lamkanfi, "Happy birthday! A trend analysis on past MSR papers," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 353–362.

[10] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, Mar. 2003.

[12] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571 – 583, 2007.