UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

UNIVERSITÄT
BERN

CiteWise

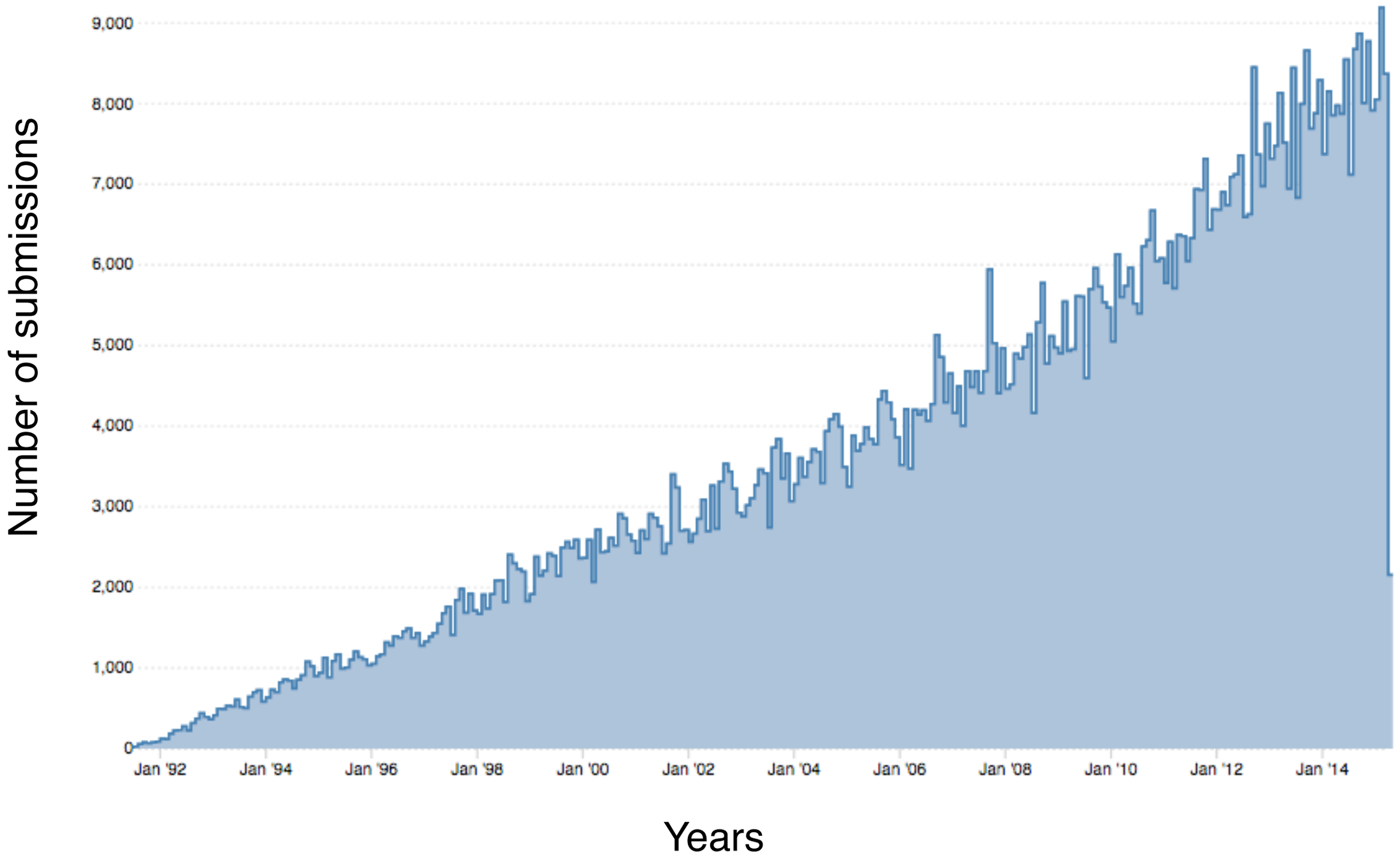
citation search engine

Aliya Ibragimova

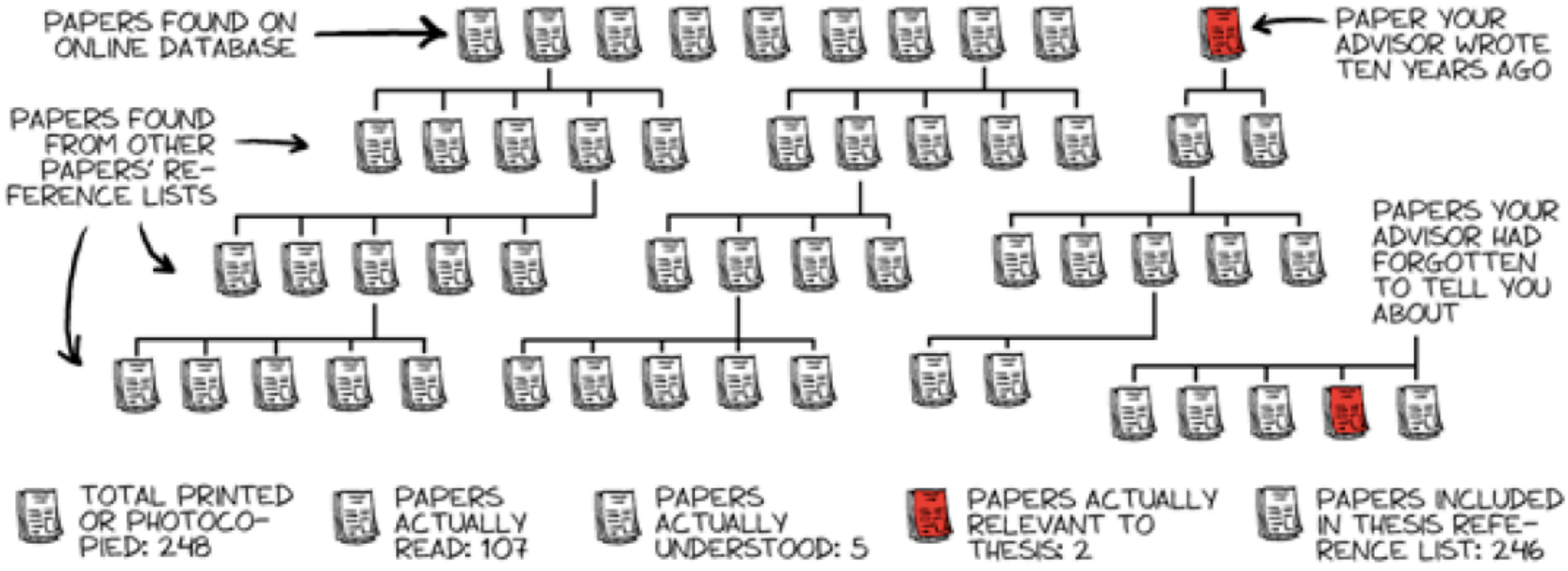
Supervisors:

Prof. Dr. Oscar Nierstrasz
Haidar Osman, Boris Spasojević

ArXiv: monthly submission rates



References: making sure no one has already written your thesis

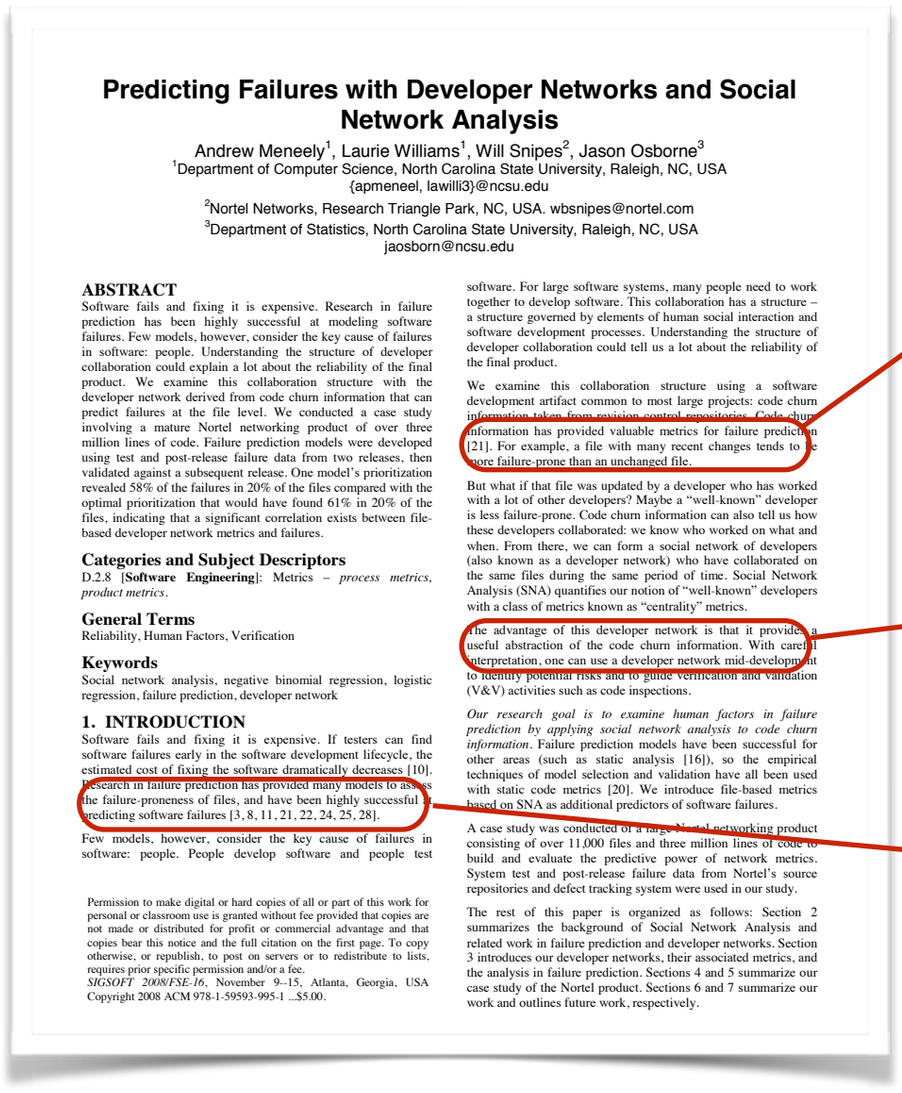


www.phdcomics.com

Why finding related work is so difficult?

- It's difficult to validate proposed claims
- look for what other people used as references to their claims

CiteWise



Predicting Failures with Developer Networks and Social Network Analysis

Andrew Meneely[1], Laurie Williams[1], Will Snipes[2], Jason Osborne[3]
[1]Department of Computer Science, North Carolina State University, Raleigh, NC, USA
{apmeneel, lawilli3}@ncsu.edu
[2]Nortel Networks, Research Triangle Park, NC, USA. wbsnipes@nortel.com
[3]Department of Statistics, North Carolina State University, Raleigh, NC, USA
jaosborn@ncsu.edu

ABSTRACT
Software fails and fixing it is expensive. Research in failure prediction has been highly successful at modeling software failures. Few models, however, consider the key cause of failures in software: people. Understanding the structure of developer collaboration could explain a lot about the reliability of the final product. We examine this collaboration structure with the developer network derived from code churn information that can predict failures at the file level. We conducted a case study involving a mature Nortel networking product of over three million lines of code. Failure prediction models were developed using test and post-release failure data from two releases, then validated against a subsequent release. One model's prioritization revealed 58% of the failures in 20% of the files compared with the optimal prioritization that would have found 61% in 20% of the files, indicating that a significant correlation exists between file-based developer network metrics and failures.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics*.

General Terms
Reliability, Human Factors, Verification

Keywords
Social network analysis, negative binomial regression, logistic regression, failure prediction, developer network

1. INTRODUCTION
Software fails and fixing it is expensive. If testers can find software failures early in the software development lifecycle, the estimated cost of fixing the software dramatically decreases [10]. Research in failure prediction has provided many models to assess the failure-proneness of files, and have been highly successful at predicting software failures [3, 8, 11, 21, 22, 24, 25, 28].

Few models, however, consider the key cause of failures in software: people. People develop software and people test software. For large software systems, many people need to work together to develop software. This collaboration has a structure – a structure governed by elements of human social interaction and software development processes. Understanding the structure of developer collaboration could tell us a lot about the reliability of the final product.

We examine this collaboration structure using a software development artifact common to most large projects: code churn information taken from revision control repositories. Code churn information has provided valuable metrics for failure prediction [21]. For example, a file with many recent changes tends to be more failure-prone than an unchanged file.

But what if that file was updated by a developer who has worked with a lot of other developers? Maybe a "well-known" developer is less failure-prone. Code churn information can also tell us how these developers collaborated: we know who worked on what and when. From there, we can form a social network of developers (also known as a developer network) who have collaborated on the same files during the same period of time. Social Network Analysis (SNA) quantifies our notion of "well-known" developers with a class of metrics known as "centrality" metrics.

The advantage of this developer network is that it provides a useful abstraction of the code churn information. With careful interpretation, one can use a developer network mid-development to identify potential risks and to guide verification and validation (V&V) activities such as code inspections.

Our research goal is to examine human factors in failure prediction by applying social network analysis to code churn information. Failure prediction models have been successful for other areas (such as static analysis [16]), so the empirical techniques of model selection and validation have all been used with static code metrics [20]. We introduce file-based metrics based on SNA as additional predictors of software failures.

A case study was conducted of a large Nortel networking product consisting of over 11,000 files and three million lines of code to build and evaluate the predictive power of network metrics. System test and post-release failure data from Nortel's source repositories and defect tracking system were used in our study.

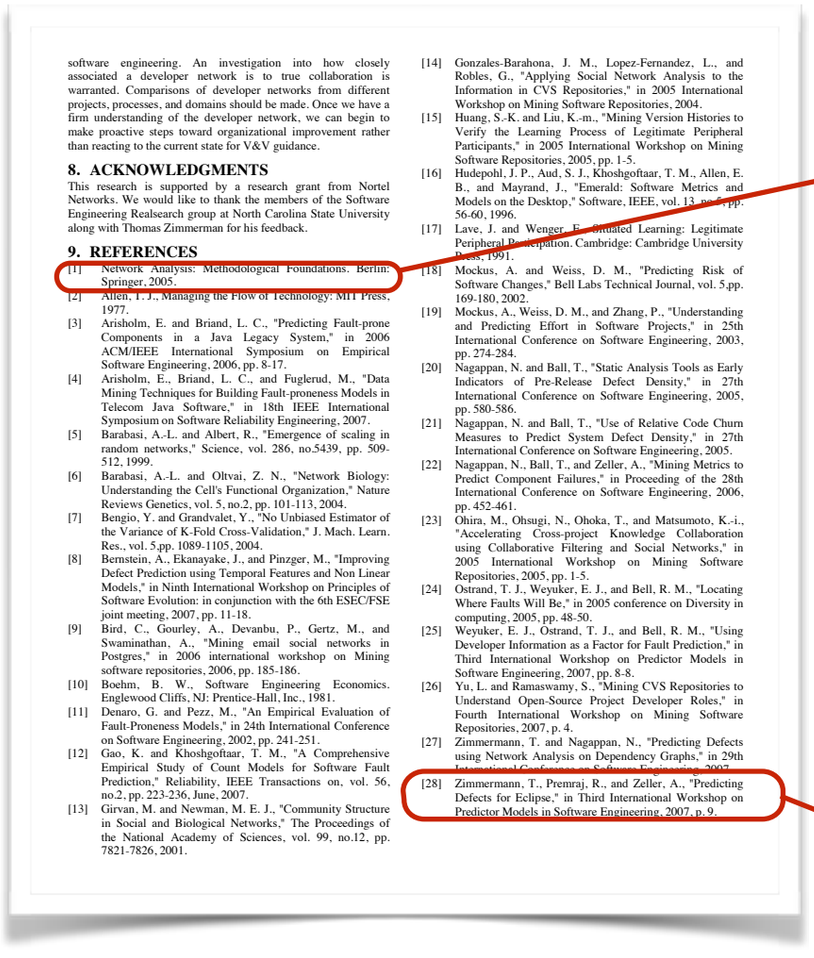
The rest of this paper is organized as follows: Section 2 summarizes the background of Social Network Analysis and related work in failure prediction and developer networks. Section 3 introduces our developer networks, their associated metrics, and the analysis in failure prediction. Sections 4 and 5 summarize our case study of the Nortel product. Sections 6 and 7 summarize our work and outlines future work, respectively.

Code churn information has provided valuable metrics for failure prediction **[21]**.

Failure prediction models have been successful for other areas (such as static analysis **[16]**), so the empirical techniques of model selection and validation have all been used with static code metrics **[20]**.

Research in failure prediction has provided many models to assess the failure-proneness of files, and have been highly successful at predicting software failures **[3, 8, 11, 21, 22, 24, 25, 28]**.

CiteWise



software engineering. An investigation into how closely associated a developer network is to true collaboration is warranted. Comparisons of developer networks from different projects, processes, and domains should be made. Once we have a firm understanding of the developer network, we can begin to make proactive steps toward organizational improvement rather than reacting to the current state for V&V guidance.

8. ACKNOWLEDGMENTS

This research is supported by a research grant from Nortel Networks. We would like to thank the members of the Software Engineering Realsearch group at North Carolina State University along with Thomas Zimmerman for his feedback.

9. REFERENCES

[1] Network Analysis: Methodological Foundations. Berlin: Springer, 2005.
[2] Allen, T. J., Managing the Flow of Technology: MIT Press, 1977.
[3] Arisholm, E. and Briand, L. C., "Predicting Fault-prone Components in a Java Legacy System," in 2006 ACM/IEEE International Symposium on Empirical Software Engineering, 2006, pp. 8-17.
[4] Arisholm, E., Briand, L. C., and Fuglerud, M., "Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software," in 18th IEEE International Symposium on Software Reliability Engineering, 2007.
[5] Barabasi, A.-L. and Albert, R., "Emergence of scaling in random networks," Science, vol. 286, no.5439, pp. 509-512, 1999.
[6] Barabasi, A.-L. and Oltvai, Z. N., "Network Biology: Understanding the Cell's Functional Organization," Nature Reviews Genetics, vol. 5, no.2, pp. 101-113, 2004.
[7] Bengio, Y. and Grandvalet, Y., "No Unbiased Estimator of the Variance of K-Fold Cross-Validation," J. Mach. Learn. Res., vol. 5,pp. 1089-1105, 2004.
[8] Bernstein, A., Ekanayake, J., and Pinzger, M., "Improving Defect Prediction using Temporal Features and Non Linear Models," in Ninth International Workshop on Principles of Software Evolution: in conjunction with the 6th ESEC/FSE joint meeting, 2007, pp. 11-18.
[9] Bird, C., Gourley, A., Devanbu, P., Gertz, M., and Swaminathan, A., "Mining email social networks in Postgres," in 2006 international workshop on Mining software repositories, 2006, pp. 185-186.
[10] Boehm, B. W., Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
[11] Denaro, G. and Pezz, M., "An Empirical Evaluation of Fault-Proneness Models," in 24th International Conference on Software Engineering, 2002, pp. 241-251.
[12] Gao, K. and Khoshgoftaar, T. M., "A Comprehensive Empirical Study of Count Models for Software Fault Prediction," Reliability, IEEE Transactions on, vol. 56, no.2, pp. 223-236, June, 2007.
[13] Girvan, M. and Newman, M. E. J., "Community Structure in Social and Biological Networks," The Proceedings of the National Academy of Sciences, vol. 99, no.12, pp. 7821-7826, 2001.
[14] Gonzales-Barahona, J. M., Lopez-Fernandez, L., and Robles, G., "Applying Social Network Analysis to the Information in CVS Repositories," in 2005 International Workshop on Mining Software Repositories, 2004.
[15] Huang, S.-K. and Liu, K.-m., "Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants," in 2005 International Workshop on Mining Software Repositories, 2005, pp. 1-5.
[16] Hudepohl, J. P., Aud, S. J., Khoshgoftaar, T. M., Allen, E. B., and Mayrand, J., "Emerald: Software Metrics and Models on the Desktop," Software, IEEE, vol. 13, no.5, pp. 56-60, 1996.
[17] Lave, J. and Wenger, E., Situated Learning: Legitimate Peripheral Participation. Cambridge: Cambridge University Press, 1991.
[18] Mockus, A. and Weiss, D. M., "Predicting Risk of Software Changes," Bell Labs Technical Journal, vol. 5,pp. 169-180, 2002.
[19] Mockus, A., Weiss, D. M., and Zhang, P., "Understanding and Predicting Effort in Software Projects," in 25th International Conference on Software Engineering, 2003, pp. 274-284.
[20] Nagappan, N. and Ball, T., "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," in 27th International Conference on Software Engineering, 2005, pp. 580-586.
[21] Nagappan, N. and Ball, T., "Use of Relative Code Churn Measures to Predict System Defect Density," in 27th International Conference on Software Engineering, 2005.
[22] Nagappan, N., Ball, T., and Zeller, A., "Mining Metrics to Predict Component Failures," in Proceeding of the 28th International Conference on Software Engineering, 2006, pp. 452-461.
[23] Ohira, M., Ohsugi, N., Ohoka, T., and Matsumoto, K.-i., "Accelerating Cross-project Knowledge Collaboration using Collaborative Filtering and Social Networks," in 2005 International Workshop on Mining Software Repositories, 2005, pp. 1-5.
[24] Ostrand, T. J., Weyuker, E. J., and Bell, R. M., "Locating Where Faults Will Be," in 2005 conference on Diversity in computing, 2005, pp. 48-50.
[25] Weyuker, E. J., Ostrand, T. J., and Bell, R. M., "Using Developer Information as a Factor for Fault Prediction," in Third International Workshop on Predictor Models in Software Engineering, 2007, pp. 8-8.
[26] Yu, L. and Ramaswamy, S., "Mining CVS Repositories to Understand Open-Source Project Developer Roles," in Fourth International Workshop on Mining Software Repositories, 2007, p. 4.
[27] Zimmermann, T. and Nagappan, N., "Predicting Defects using Network Analysis on Dependency Graphs," in 29th International Conference on Software Engineering, 2007.
[28] Zimmermann, T., Premraj, R., and Zeller, A., "Predicting Defects for Eclipse," in Third International Workshop on Predictor Models in Software Engineering, 2007, p. 9.

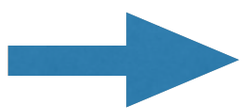
[1] Arisholm, E. and Briand, L. C., "Predicting Fault-prone Components in a Java Legacy System," in 2006 ACM/IEEE International Symposium on Empirical Software Engineering, 2006, pp. 8-17.

-
-
-

[28] Zimmermann, T., Premraj, R., and Zeller, A., "Predicting Defects for Eclipse," in Third International Workshop on Predictor Models in Software Engineering, 2007, p. 9.

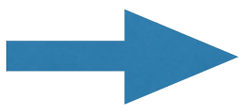
CiteWise

Code churn information has provided valuable metrics for failure prediction **[21]**.



[21] Nagappan, N. and Ball, T., "Use of Relative Code Churn Measures to Predict System Defect Density," in 27th International Conference on Software Engineering, 2005.

Research in failure prediction has provided many models to assess the failure-proneness of files, and have been highly successful at predicting software failures **[3, 8, 11, 21, 22, 24, 25, 28]**.



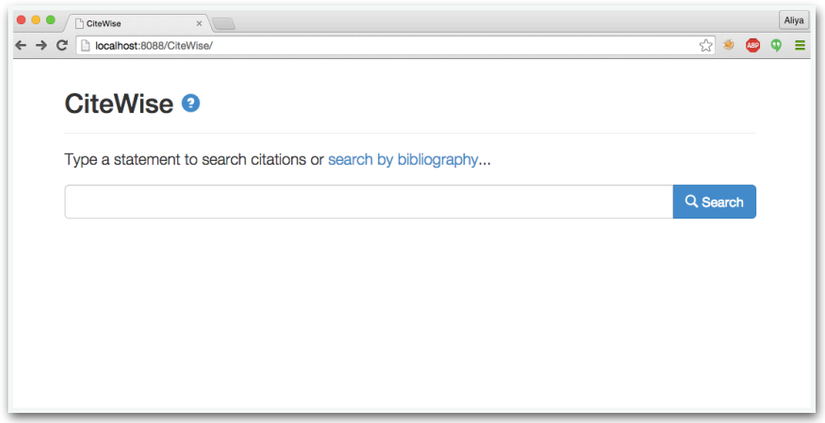
[3] Arisholm, E. and Briand, L. C., "Predicting Fault-prone Components in a Java Legacy System," in 2006 ACM/IEEE International Symposium on Empirical Software Engineering, 2006, pp. 8-17.

•
•
•

[21] Nagappan, N. and Ball, T., "Use of Relative Code Churn Measures to Predict System Defect Density," in 27th International Conference on Software Engineering, 2005.

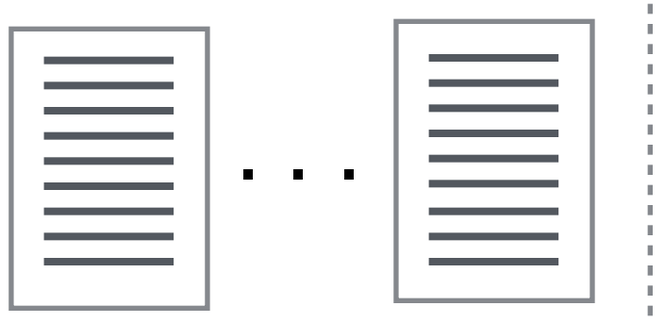
CiteWise

DB



Parser

Indexer



Experiments setup

- 16 000 articles in Programming Languages and Software Engineering
- 9 participants
- 3 different tasks:
  - Task 1a and Task 1b - for comparing CiteWise with Google Scholar
  - Task 2 - for comparing summaries of CiteWise with TextRank algorithm

Task 1a

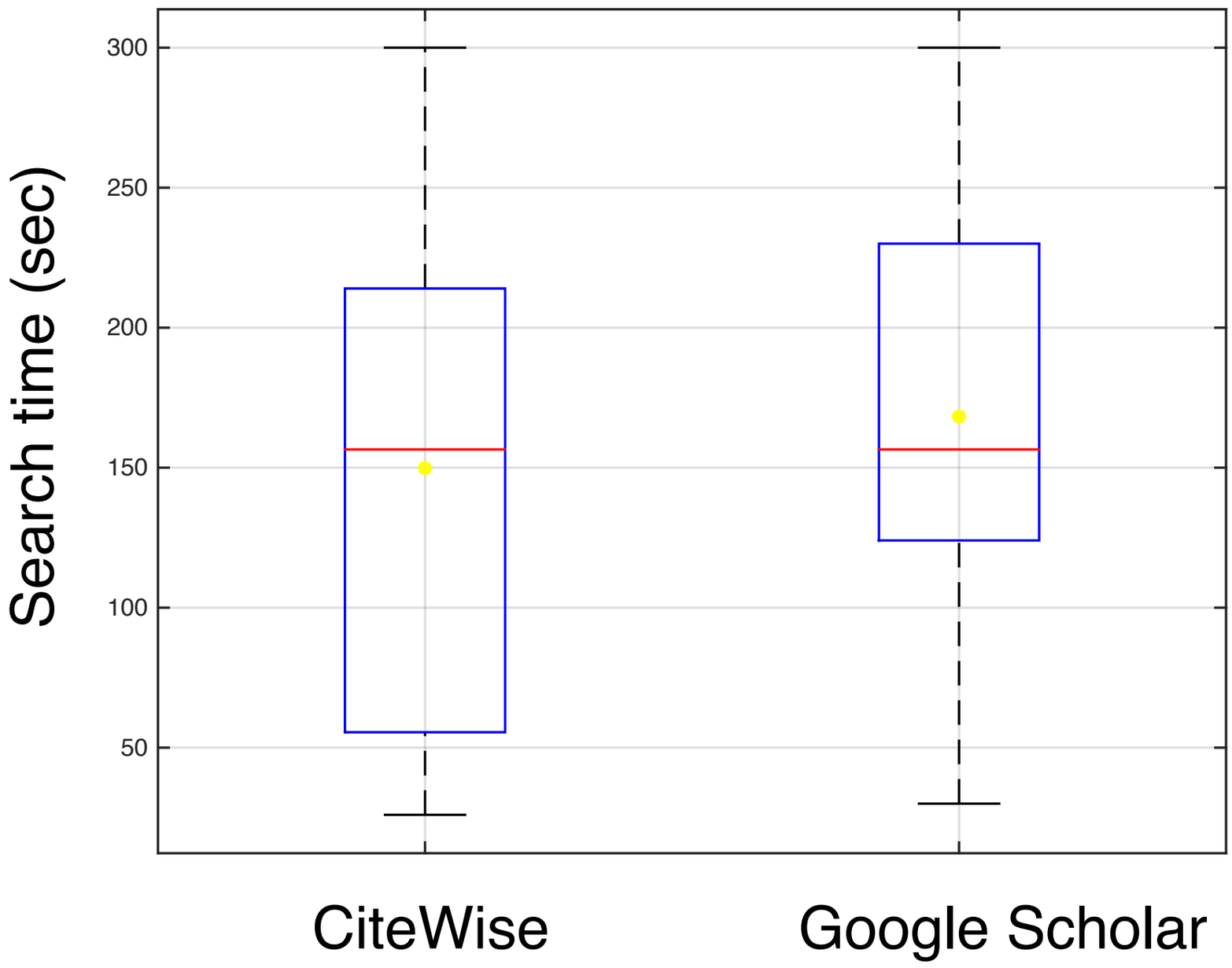
Description: find a reference to the claim using **CiteWise** or **Google Scholar**

Goal: find out which search engine is more efficient

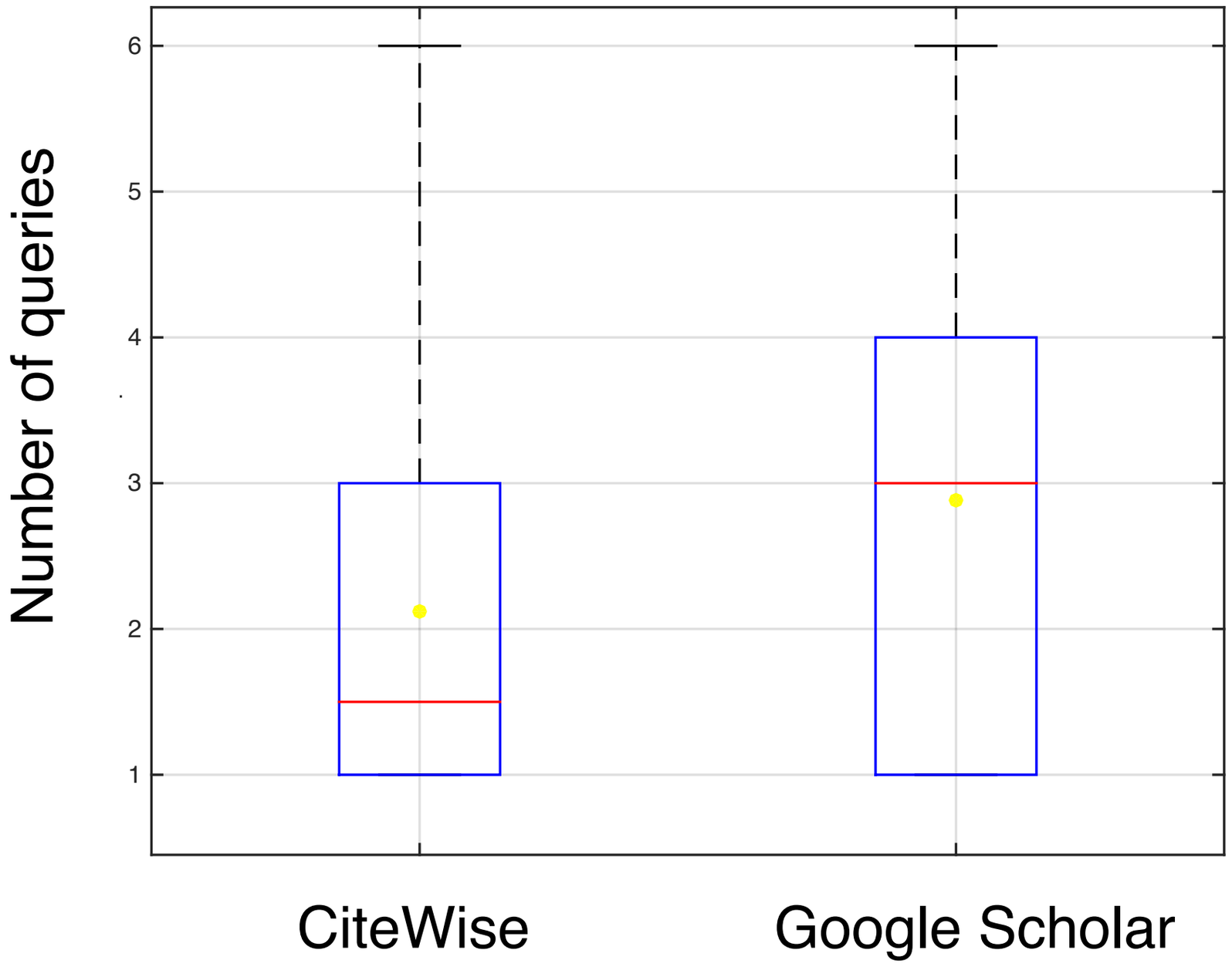
Observed variables:

- search time
- number of queries
- average number of words in a query

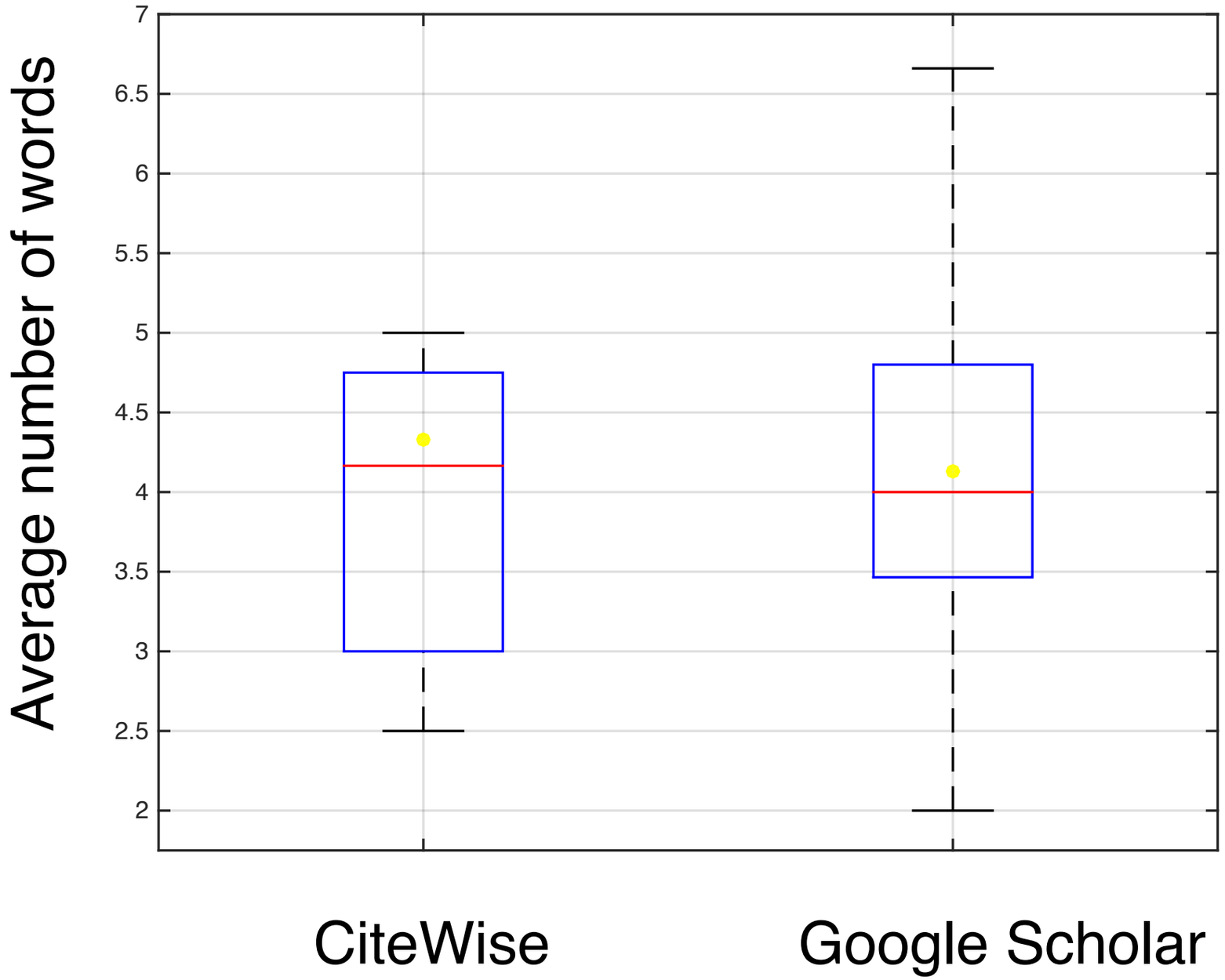
Search time in Task1a



Number of queries in Task1a



Average number of words in Task1a



Task 1b

Description: find a reference to the claim using either **Google Scholar** or **CiteWise**

Goal: find out user preferences in using search engines

Observed variables:

- search engine where the result was retrieved
- if a participant tried to use both search engines

Results for Task1b

Participants	How SEs were used
P6	CW, GS, CW
P7	GS, CW
P8	GS, CW
P9	CW

CW - CiteWise GS - Google Scholar

Task 2

Description: compare two summaries generated with one of the popular summary generation algorithms (**TextRank***)* and **CiteWise**

Goal: find out which summary is better

Observed variables:

- scores given to summaries by participants

Results for Task 2

Results summary

- CiteWise was slightly better for search time and number of queries, however no statistically significant difference
- In 50% of cases for CiteWise in Task 1a, participants found a reference in less than 2 queries
- All users succeeded with CiteWise in Task 1b
- Summaries generated with CiteWise were better

Contributions

- A novel IR system for scientific articles based on citations
- A new method of summary generation by means of citation aggregation
- An empirical evaluation of the system by means of user study experiments

Appendix

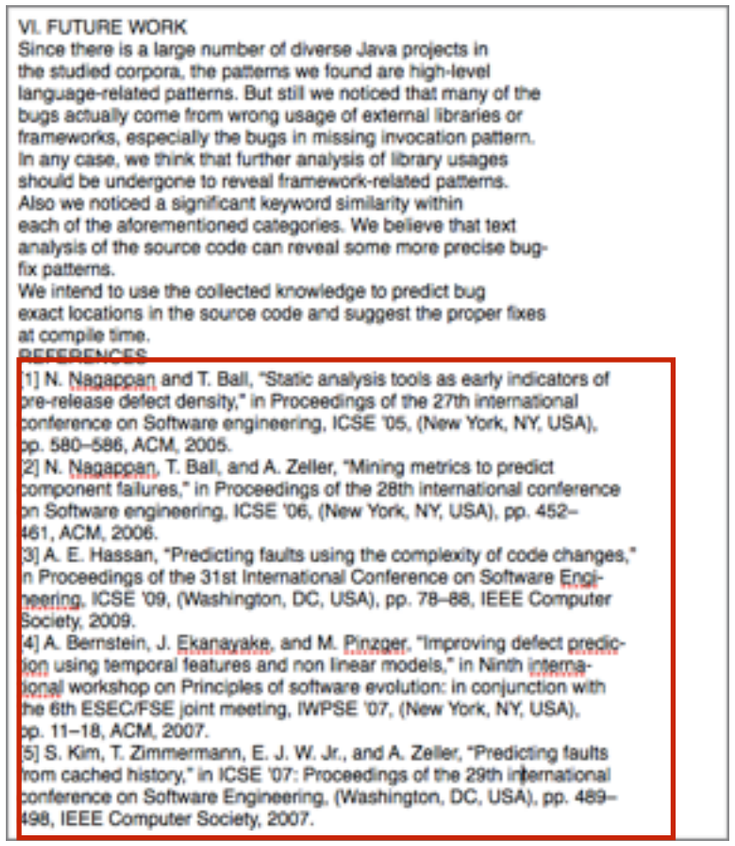
Parser

- Extract citations
- Match citations with references
- Aggregate citations referring to the same source paper

Extracting citations

1. PDF document -> plain text

2. Breaking down a document using keywords



VI. FUTURE WORK
Since there is a large number of diverse Java projects in the studied corpora, the patterns we found are high-level language-related patterns. But still we noticed that many of the bugs actually come from wrong usage of external libraries or frameworks, especially the bugs in missing invocation pattern. In any case, we think that further analysis of library usages should be undergone to reveal framework-related patterns. Also we noticed a significant keyword similarity within each of the aforementioned categories. We believe that text analysis of the source code can reveal some more precise bug-fix patterns.
We intend to use the collected knowledge to predict bug exact locations in the source code and suggest the proper fixes at compile time.
REFERENCES
[1] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in Proceedings of the 27th international conference on Software engineering, ICSE '05, (New York, NY, USA), pp. 580–586, ACM, 2005.
[2] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in Proceedings of the 28th international conference on Software engineering, ICSE '06, (New York, NY, USA), pp. 452–461, ACM, 2006.
[3] A. E. Hassan, "Predicting faults using the complexity of code changes," in Proceedings of the 31st International Conference on Software Engineering, ICSE '09, (Washington, DC, USA), pp. 78–88, IEEE Computer Society, 2009.
[4] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, IWPSE '07, (New York, NY, USA), pp. 11–18, ACM, 2007.
[5] S. Kim, T. Zimmermann, E. J. W. Jr., and A. Zeller, "Predicting faults from cached history," in ICSE '07: Proceedings of the 29th international conference on Software Engineering, (Washington, DC, USA), pp. 489–498, IEEE Computer Society, 2007.

*{REFERENCES, References, …}

•

•

•

*{APPENDIX, Appendix, …}

3. Text normalisation

   - replace dashes in case of words splitting
   - replace new lines with white spaces in case of sent. splitting

Extracting citations

4. Break document 'body' into sentences

5. Extract sentences having references:

   [12], [12, 15, 32], [6, p.35], [Alu95]

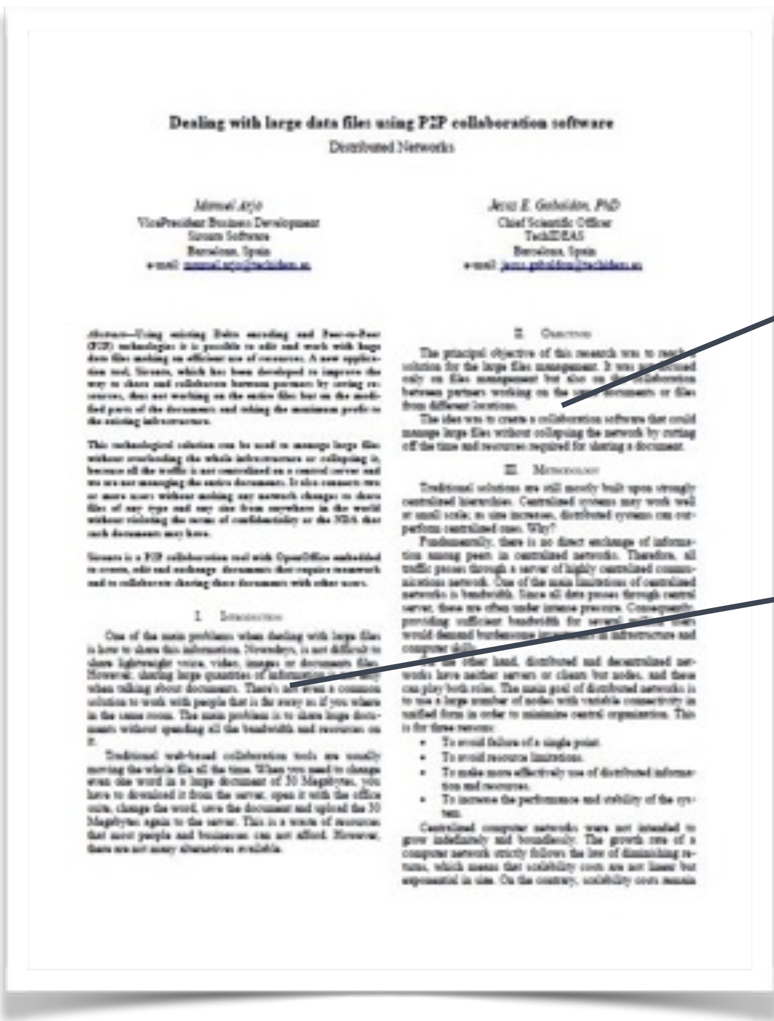
6. Extract references

   [12] N. Nagappan and T. Ball, "Static analysis tools …

7. Match sentences with references

8. Publish to the Indexer

Aggregating citations


Dealing with large data files using P2P collaboration software

Distributed Networks

> Besides, Arisholm *et al*. found that the cost-effectiveness of bug prediction based on source code metrics is actually close to zero **[30]** due to the correlation with the size.

> Also a good way to evaluate the prediction model would be a cost-efficiency evaluation where the cost is the LOC, since the costs of unit tests and code reviews are approximately proportional to the size of the source code file **[30]**.

•

•

[30] E. Arisholm, L. C. Briand, and E. B. Johannessen, "**A systematic and comprehensive investigation of methods to build and evaluate fault prediction models**," *J. Syst. Softw.*, vol. 83, pp. 2–17, Jan. 2010.

Aggregating citations

Conradi, R., Dyba, T., Sjoberg, D.I.K., and Ulsund, T., "Lessons learned and recommendations from two large norwegian SPI programmes." Lecture notes in computer science, 2003, pp. 32-45."

P. Molin, L. Ohlsson, 'Points & Deviations - A pattern language for fire alarm systems,' to be published in Pattern Languages of Program Design 3, Addison- Wesley.

Title is a text enclosed in quotes, e.g " or '

Aggregating citations

Authors	Title	Rest
R. P. Wilson and M. S. Lam.	Effective context sensitive pointer analysis for C programs	In PLDI, pages 1–12, June 1995. 289

- Break reference into logical parts using NLP library, take the second part
- Normalise, remove punctuation marks, lowercase

Open issues

1. 

The mistake-counting model that we use is essentially the same as a model discussed in Barzdin and Freivald (1972). See Angluin and Smith (1983) for a survey that compares a number of learning models.

2. 

epre.Counter.bump() ≡ [τ = ϙ]
epost.Counter.bump() ≡
  [(this.lstnr 6= null) ⇒
    ((|τ| = 1)
    ∧ (τ[1].hm
    = this.lstnr.actionPerformed))]
 ∧ [(this.lstnr = null) ⇒ τ = ϙ]