

# Concept-Based Exploration of Rich Semi-Structured Data Collections

by

Gillian J. Greene



*Dissertation presented for the degree of Doctor of Philosophy in the  
Faculty of Science at Stellenbosch University*

Supervisor: Prof. Bernd Fischer

March 2017

# Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ..... March 2017 .....

Copyright © 2017 Stellenbosch University  
All rights reserved.

# Abstract

## Concept-Based Exploration of Rich Semi-Structured Data Collections

G.J. Greene

*Division of Computer Science,*

*University of Stellenbosch,*

*Private Bag X1, Matieland 7602, South Africa.*

Dissertation: PhD

March 2017

Search has become one of the fundamental operations in computer science, allowing users to extract data and ultimately information from datasets. However, when users have no previous knowledge of a dataset, or have not clearly defined their search task and are therefore unable to formulate a direct query, their task becomes one of *exploratory search* or *browsing* rather than focused search or retrieval. While search and retrieval tools are widely provided, support for browsing of large and especially rich semi-structured datasets, is lacking. Semi-structured data is particularly difficult to explore and query because treating it as complete free-text causes a loss of important additional information which is encoded in the structured portions of the data while considering only the structured fields results in the loss of important free-text information. We therefore develop a framework to support exploration of semi-structured data, which is otherwise difficult to gather insights from, without requiring the user to have prior knowledge of the dataset or have formulated a specific query. Our approach uses a novel combination of tag clouds and concept lattices to facilitate data exploration, analysis and visualization by allowing the user to continuously update (i.e., add and remove) a set of keywords that the searched documents must contain. The document set is not directly provided as the result of a specific query, but aggregated information and properties of relevant documents are provided as a result. We apply our framework to data contained in software repositories, in two different ways for different goals to highlight the flexibility of the approach and the different tasks that can be supported using the same

*ABSTRACT*

iii

underlying dataset. We also instantiate our framework to support the exploration of a large collection of academic publication data. We evaluate the instantiations of our framework by conducting user and case studies, which indicate that our approach is usable and allows users to gather valuable information from semi-structured data archives.

# Uittreksel

## Konsep-gebaseerde verkenning van ryk semi-gestruktureerde data-versamelings

G.J. Greene

*Afdeling Rekenaarwetenskap ,  
Universiteit van Stellenbosch,  
Privaatsak X1, Matieland 7602, Suid Afrika.*

Proefskrif: PhD

Maart 2017

Soektogte is een van die fundamentele operasies in rekenaarwetenskap. Dit laat gebruikers toe om data, en uiteindelik inligting, vanuit datastelle te onttrek. Wanneer gebruikers egter geen vorige kennis van 'n datastel het nie, of hul soektog nie duidelik gedefinieer het nie, en dus nie in staat is om 'n direkte navraag te formuleer nie, word hul taak een van verkennende soek, of blaai, eerder as gefokusde soek of herwinning. Terwyl soeken herwinnings-instrumente algemeen beskikbaar is, ontbreek ondersteuning vir die verkenning van groot en veral ryk semi-gestruktureerde datastelle. Semi-gestruktureerde data is veral moeilik om te verken en na te vra omdat die hantering daarvan as slegs vrye teks 'n verlies van belangrike aanvullende inligting veroorsaak wat ingebou is in die gestruktureerde gedeeltes van die data, terwyl die inagneming van slegs die gestruktureerde velde weer lei tot 'n verlies van belangrike vrye teks inligting. Ons ontwikkel dus 'n raamwerk om die verkenning van semi-gestruktureerde data te ondersteun, wat andersins moeilik is om insigte uit te verkry, sonder om van die gebruiker te vereis dat hulle voorafgaande kennis van die datastel het, of 'n spesifieke navraag reeds geformuleer het. Ons benadering maak gebruik van 'n nuwe kombinasie van etiket-wolke en konsep-roosters om data-verkenning, data-analise, en data-visualisering te fasiliteer deur die gebruiker toe te laat om voortdurend 'n stel sleutelwoorde op te dateer (m.a.w. by te voeg of te verwyder) wat bevat moet word in die dokumente waarvoor gesoek word. Die dokument-stel word nie direk verskaf as die resultaat van 'n spesifieke navraag nie,

maar saamgestelde inligting en eienskappe van relevante dokumente word eerder verskaf. Ons pas ons raamwerk toe op data wat in programmatuurargiewe gestoor is op twee verskillende maniere vir verskillende doelwitte om die buigsaamheid van die benadering en die verskillende take wat ondersteun kan word met behulp van dieselfde onderliggende datastel uit te lig. Ons instansieer ook ons raamwerk om die verkenning van 'n groot versameling van akademiese publikasiedata te ondersteun. Ons evalueer die instansies van ons raamwerk deur die gebruik van gebruiker en gevallestudies, wat daarop dui dat ons benadering bruikbaar is, en gebruikers in staat stel om waardevolle inligting vanuit semi-gestruktureerde data-argiewe in te samel.

# List of Publications

- [1] G. J. Greene, B. Fischer, Conceptcloud: A tagcloud browser for software archives, in: Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, 2014, pp. 759–762.
- [2] G. J. Greene, B. Fischer, Interactive tag cloud visualization of software version control repositories, in: 3rd Working Conference on Software Visualization (VISSOFT), IEEE, 2015, pp. 56–65.
- [3] G. J. Greene, A generic framework for concept-based exploration of semi-structured software engineering data, in: Automated Software Engineering Doctoral Symposium, IEEE, 2015, pp. 894–897.
- [4] G. J. Greene, B. Fischer, Single-focus broadening navigation in concept lattices., in: International Workshop on Concept Discovery in Unstructured Data (CDUD), 2016, CEUR Workshop Proceedings.
- [5] G. J. Greene, B. Fischer, CVExplorer: Identifying Candidate Developers by Mining and Exploring their Open Source Contributions., in: Automated Software Engineering Tool Demonstrations (ASE), 2016, ACM.
- [6] M. Dunaiski, G. J. Greene, B. Fischer, Exploratory Search of Academic Publication and Citation Data using Interactive Tag Cloud Visualizations., Accepted for publication in Scientometrics, Springer.
- [7] G. J. Greene, M. Esterhuizen, B. Fischer, Visualizing and Exploring Software Version Control Repositories using Interactive Tag Clouds over Formal Concept Lattices., Accepted for publication in Information and Software Technology, Elsevier.

# Acknowledgments

Thank you to my supervisor, Bernd Fischer, for all your support, guidance and enthusiasm, which has made this research possible.

Thank you to the examiners for their valuable feedback which has been incorporated into the thesis.

Thank you to Marion, Laura, Edward, Ignatius and Fovvy for your encouragement and all that you do for me. Thank you to Miesje and Pieter for your role in everything I have done.

Thank you also to Ronald and Tarnya for your support in making this possible.

The financial assistance of the STIAS Doctoral Scholarship, CAIR-SU, CSIR, DST, MIH Media Lab and the National Research Foundation (Grant Number 93582) towards this research is gratefully acknowledged.



# Dedications

*For Pieter Hendrik Fasol*

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Uittreksel</b>	<b>iv</b>
<b>List of Publications</b>	<b>vi</b>
<b>Acknowledgments</b>	<b>vii</b>
<b>Dedications</b>	<b>viii</b>
<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Description . . . . .	3
1.2 Research Objectives . . . . .	3
1.2.1 Visual Exploration of Software Development Repositories . .	4
1.2.2 Identification of Skills from Aggregated Software Repository Data . . . . .	5
1.2.3 Exploration of Academic Publication Data . . . . .	6
1.3 Outline of the Solution . . . . .	7
1.3.1 Research Methodology . . . . .	7
1.3.2 Technical Approach . . . . .	7
1.3.2.1 Formal Concept Lattice Construction . . . . .	7
1.3.2.2 Interface Construction . . . . .	8
1.3.2.3 Support for Navigation . . . . .	9
1.3.3 ConceptCloud Browser . . . . .	10

<i>CONTENTS</i>	<b>x</b>
1.4 Contributions . . . . .	10
1.5 Organization of the Dissertation . . . . .	11
<b>2 Background</b>	<b>14</b>
2.1 Formal Concept Analysis . . . . .	14
2.1.1 Mathematical Foundations . . . . .	14
2.1.2 Visual Presentation of Data Contained in Concept Lattices . . . . .	17
2.1.3 Navigation in Formal Concept Analysis . . . . .	18
2.2 Exploratory Search . . . . .	20
2.2.1 Navigation Styles for Exploration of Datasets . . . . .	21
2.2.2 Visual Presentation of Data for Exploration . . . . .	22
2.3 Tag Clouds . . . . .	24
<b>3 Single-Focus Navigation in Concept Lattices</b>	<b>26</b>
3.1 Introduction . . . . .	26
3.2 Refinement Navigation . . . . .	28
3.2.1 Refinement Selection . . . . .	28
3.2.2 Refinement De-Selection . . . . .	28
3.3 Broadening Navigation . . . . .	29
3.4 Boolean Disjunctive Selection . . . . .	32
3.5 Attribute-Oriented Generalization . . . . .	35
3.5.1 Generating Candidate Focus Concepts . . . . .	36
3.5.2 Selecting a new Focus Concept . . . . .	37
3.5.3 Worked Example . . . . .	37
3.6 Object-Oriented Generalization: Finding Similar Objects . . . . .	40
3.7 Conclusion . . . . .	43
<b>4 ConceptCloud Tool</b>	<b>44</b>
4.1 Tag Cloud Visualization . . . . .	45
4.1.1 Tag Cloud from a Concept . . . . .	45
4.1.2 Relation to Information Retrieval . . . . .	47
4.2 Tool Architecture . . . . .	48
4.2.1 Pre-processing Steps . . . . .	50
4.2.2 Concept Lattice Construction . . . . .	50
4.3 Tag Cloud View . . . . .	51
4.3.1 Display of Multiple Viewers . . . . .	51
4.3.2 Viewer Customization . . . . .	53
4.3.3 Table Viewer . . . . .	54
4.4 ConSL: User Interface Scripting Language . . . . .	55

<i>CONTENTS</i>	<b>xi</b>
4.5 Conclusion . . . . .	56
<b>5 Concept-Based Exploration of Version Control Archives</b>	<b>58</b>
5.1 Introduction . . . . .	58
5.2 Browsing Version Control Archives . . . . .	59
5.2.1 Modeling Software Repositories . . . . .	59
5.3 Contexts from Repositories . . . . .	60
5.3.1 Basic Preprocessing . . . . .	61
5.3.2 Revision-Based Contexts . . . . .	61
5.3.3 File-Based Contexts . . . . .	62
5.3.4 Change-Based Contexts . . . . .	63
5.3.5 Combined Contexts: Bug Reports and Revision Control Data	64
5.4 Advanced Customizations in ConceptCloud . . . . .	65
5.4.1 Personalization in Tag Clouds . . . . .	66
5.4.2 Filtering Tag Clouds . . . . .	66
5.4.3 Customized Visualizations . . . . .	66
5.5 Illustrative Case Studies . . . . .	67
5.5.1 JUnit Repository . . . . .	68
5.5.1.1 Overview . . . . .	68
5.5.1.2 Authors by Month . . . . .	68
5.5.1.3 Conclusions . . . . .	69
5.5.2 RubyGems Repository . . . . .	69
5.5.2.1 Conclusions . . . . .	70
5.5.3 Industrial Case Study: Ruby on Rails Web Application . . .	72
5.5.3.1 Project Activity . . . . .	72
5.5.3.2 Developer Activity . . . . .	73
5.5.3.3 Developer Expertise . . . . .	74
5.5.3.4 Conclusions . . . . .	75
5.5.4 Industrial Case Study: Mobile Application . . . . .	75
5.5.4.1 Project Contributors . . . . .	76
5.5.4.2 Developer Collaboration . . . . .	77
5.5.4.3 Commit Activity . . . . .	79
5.5.4.4 Commit Messages . . . . .	79
5.5.5 Conclusions . . . . .	80
5.6 Performance Evaluation . . . . .	80
5.7 User Study . . . . .	82
5.7.1 Experimental Setup . . . . .	83
5.7.2 Population . . . . .	83
5.7.3 Tasks . . . . .	83

<i>CONTENTS</i>	<b>xii</b>
5.7.4 Analysis and Results . . . . .	85
5.7.4.1 Summary Statistics . . . . .	85
5.7.4.2 Statistical Significance . . . . .	88
5.7.5 Question Types . . . . .	89
5.7.6 Discussion . . . . .	90
5.7.7 Threats to Validity . . . . .	90
5.8 Related Work . . . . .	91
5.8.1 Visualizing Software and Bug Repositories . . . . .	91
5.8.2 Tag Cloud Visualizations of Software . . . . .	92
5.8.3 Software Engineering and Formal Concept Analysis . . . . .	93
5.9 Conclusion . . . . .	93
<b>6 Concept-Based Exploration of Developer Skill Sets Identified from Their Open Source Contributions</b>	<b>95</b>
6.1 Introduction . . . . .	95
6.2 Background . . . . .	98
6.2.1 Candidate Identification . . . . .	98
6.2.2 Interpreting User Profiles . . . . .	98
6.2.3 Business-Oriented Social Networks . . . . .	99
6.2.4 Open Source Hosting Sites . . . . .	99
6.2.5 Profile and Skill Aggregators . . . . .	99
6.3 Role of Open Source Contributions in Hiring Technical Candidates .	101
6.3.1 Study Design . . . . .	101
6.3.1.1 GitHub Users . . . . .	101
6.3.1.2 Recruiters . . . . .	102
6.3.2 GitHub User Survey Findings . . . . .	103
6.3.2.1 Recruiter Contact . . . . .	103
6.3.2.2 Other Social Platforms . . . . .	103
6.3.2.3 Developer Sentiment on Contact From Recruiters .	105
6.3.2.4 Reasons for Providing Contact Information on Profile	106
6.3.2.5 Role of GitHub in Respondent's Past and Current Employment . . . . .	106
6.3.3 Recruiter Survey and Interview Findings . . . . .	107
6.3.4 Survey Summary . . . . .	107
6.3.4.1 GitHub Users . . . . .	107
6.3.4.2 Recruiters . . . . .	107
6.3.4.3 Technical Aspects . . . . .	108
6.3.4.4 GitHub in the Hiring Process . . . . .	108
6.4 Presenting Developer Contributions in a Browsable Format . . . . .	108

6.4.1	Evaluating Skills via GitHub's Interface . . . . .	109
6.4.2	Mining Developer Contributions . . . . .	110
6.4.3	Aggregating Developer Contributions . . . . .	110
6.4.4	Presenting Developer Contributions in a Browsable Format . . . . .	111
6.5	Comparing Open-Source CVs to Other Developer Profile Data . . . . .	113
6.5.1	Comparison to LinkedIn Profile Data . . . . .	113
6.5.2	Comparison to Stack Overflow Profile Data . . . . .	116
6.5.3	Summary . . . . .	117
6.6	Identifying Candidates for Open Positions . . . . .	117
6.6.1	Suitability of Candidates for Company A . . . . .	118
6.6.2	Suitability of Candidates for Company B . . . . .	118
6.6.3	Summary . . . . .	118
6.7	Related Work . . . . .	119
6.7.1	Mining GitHub User Data for Employment Activities . . . . .	119
6.7.2	Social Recruiting . . . . .	120
6.7.3	Assessment of Developers by Open Source Developers . . . . .	120
6.8	Conclusions . . . . .	121
<b>7</b>	<b>Concept-Based Exploration of Academic Publication and Citation</b>	
	<b>Data</b>	<b>122</b>
7.1	Introduction . . . . .	122
7.2	Constructing Contexts from Publication Data . . . . .	124
7.2.1	Paper-Based Contexts . . . . .	124
7.2.2	Key-Phrase Extraction . . . . .	125
7.3	ConceptCloud Publication Browser . . . . .	126
7.4	Handling Object References with Formal Concept Lattices . . . . .	128
7.5	Worked Examples . . . . .	130
7.5.1	Prominent Authors . . . . .	130
7.5.2	Author Collaboration . . . . .	131
7.5.3	Topic Trends . . . . .	131
7.5.4	Citation Analysis . . . . .	132
7.6	User Study . . . . .	132
7.6.1	Question Set . . . . .	134
7.6.2	Population . . . . .	135
7.6.3	Pilot Study . . . . .	137
7.6.4	Results . . . . .	139
7.6.4.1	Time Taken per Question . . . . .	139
7.6.4.2	Correctness of Answers . . . . .	140
7.6.4.3	Results per Question . . . . .	140

<i>CONTENTS</i>	<b>xiv</b>
7.6.4.4 Participant Feedback . . . . .	143
7.6.5 Analysis of Results . . . . .	144
7.6.5.1 Effect of participant's background on correctness re- sults achieved . . . . .	144
7.6.5.2 Effect of use of a specific tool function on correctness results achieved . . . . .	145
7.6.6 Threats to Validity . . . . .	145
7.7 Related Work . . . . .	146
7.8 Conclusions . . . . .	147
<b>8 Conclusions</b>	<b>149</b>
8.1 Conclusions . . . . .	149
8.2 Further Work Directions . . . . .	151
8.2.1 Generic Framework . . . . .	152
8.2.2 Software Development Archive Browsing . . . . .	153
8.2.3 Skills Browsing . . . . .	153
8.2.4 Publication Browsing . . . . .	154
8.2.5 Additional Applications . . . . .	154
8.3 Final Remarks . . . . .	154
<b>Appendices</b>	<b>155</b>
<b>A ConSL Grammar</b>	<b>156</b>
<b>List of References</b>	<b>157</b>

# List of Figures

2.1	Example context derived from wine review data. . . . .	15
2.2	Example concept lattice derived from wine review data . . . . .	16
2.3	A depiction of a large concept lattice. . . . .	18
2.4	The Tag Trails Interface . . . . .	23
2.5	The Film Finder Interface . . . . .	23
2.6	The Stuff I've Seen Interface . . . . .	24
2.7	Example of a tag cloud. . . . .	25
3.1	Illustration of the meet operation. . . . .	29
3.2	Illustration of the de-selection operation. . . . .	30
3.3	Illustration of Boolean OR navigation. . . . .	34
3.4	Example of a Query concept added to a lattice . . . . .	35
3.5	Extended wine context. . . . .	38
4.1	Concepts used to construct a tag cloud from a particular focus concept .	46
4.2	Navigating concept lattices with tag clouds . . . . .	47
4.3	Main components of the ConceptCloud browser . . . . .	49
4.4	Initial tag cloud for the JUnit repository . . . . .	52
4.5	A multi-faceted tag cloud view in ConceptCloud . . . . .	52
4.6	Example of a sticky tag viewer . . . . .	53
4.7	JavaScript classes controlling different viewers in ConceptCloud . . . . .	54
4.8	Available viewer customization options in ConceptCloud . . . . .	55
5.1	Vacation cloud for David Saff . . . . .	67
5.2	The Author File Graph from a section of JUnit . . . . .	68
5.3	JUnit author clouds and file selections . . . . .	69
5.4	RubyGems: Tag cloud for commit 3642 . . . . .	70
5.5	Overview of RubyGems combined context on Gem Install selection . . .	71
5.6	Rubygems files and committers closing issues mentioning Gem Install . .	72
5.7	Dates of all commits to industrial project . . . . .	73



5.8	Weekdays of developer PP . . . . .	73
5.9	Developers of industrial project . . . . .	74
5.10	Directory cloud of developer DR . . . . .	75
5.11	Tag cloud after selection of tag Video . . . . .	76
5.12	Developer contributors over project from project start to first release . .	77
5.13	Collaboration with developer LS . . . . .	78
5.14	Directory view of developer collaborations . . . . .	78
5.15	Weekdays of developer commits. . . . .	79
5.16	Popular keywords from commit messages. . . . .	80
5.17	Average percentage obtained by participants across all three tasks . . . .	86
5.18	Box and whisker plots for average percentages obtained . . . . .	87
5.19	Boxplots of percentages obtained by participants . . . . .	87
5.20	Confidence intervals obtained from Tukey Test. . . . .	89
6.1	Tags associated with Python file changes . . . . .	97
6.2	Example of a LinkedIn profile . . . . .	100
6.3	200 largest tags from commits in which <code>.java</code> files have been edited . . .	109
6.4	Overview of skills extraction and visualization . . . . .	112
6.5	Selection of skill and developer tag . . . . .	114
6.6	Skills across Stack Overflow, LinkedIn and GitHub . . . . .	115
7.1	Main, reference and citation clouds for Model Checking . . . . .	124
7.2	Overview of the process of creating a tag cloud from publication data . .	125
7.3	Initial view of the ConceptCloud Publication Browser. . . . .	127
7.4	Table view of publications in the ConceptCloud Publication Browser . .	127
7.5	Overview of reference context creation for author Ben . . . . .	129
7.6	Top 25 prominent authors on Model Checking . . . . .	130
7.7	Author collaboration cloud of Robert Sedgewick . . . . .	131
7.8	Key-phrases of papers in International Conference on Software Engineering	132
7.9	User study questions for publication browser . . . . .	136
7.10	Population characteristics . . . . .	137
7.11	An example of the search functionality provided by ConceptCloud . . .	138
7.12	Average time taken per question in user study . . . . .	139
7.13	Correctness Percentages Achieved . . . . .	141
7.14	Average correctness for questions in user study . . . . .	141
7.15	Boxplots per individual question in user study . . . . .	142

# List of Tables

3.1	Calculated scores for candidate focus concepts . . . . .	40
3.2	Calculated $F_2$ -measures for candidate focus concepts . . . . .	40
3.3	Extent sizes for possible generalized concepts . . . . .	43
5.1	Performance metrics for revision-based contexts . . . . .	81
5.2	Question Set for User Study, a) Ruby Gems b) Backbone c) Retrofit . . . . .	85
5.3	Descriptive statistics for average percentages obtained . . . . .	86
5.4	p-values for Tukey test . . . . .	88
6.1	Survey questions posed to developers on GitHub. . . . .	102
6.2	Survey questions posed to recruiters . . . . .	104
7.1	Main functionalities provided by the publication browser grouped into categories . . . . .	133
7.2	The number of questions for each topic proposed to participants in the user study. . . . .	135
7.3	Mean and median of participant scores for the usefulness and usability of tool features. . . . .	143

# Chapter 1

## Introduction

Search has become one of the fundamental operations in computer science. Its purpose is to extract data and ultimately information from datasets. The operation itself is conceptually simple: it takes as input a query and a dataset and returns a subset of the data which matches the search query. Search engines, such as Google [14] are widely used and are continually adapted to achieve higher precision on queries; for example, Google’s personalized search takes as input a query, dataset and a user’s historical queries. Search engines also rank the results returned to the user, so while there might be a large number of results returned the most relevant results should appear first in the result list. However, other search interfaces, such as those provided by a university library, are still simplistic in nature and do not necessarily provide a ranking for the results and therefore the user is left to manually examine the full list of results that appear in the result set.

While the concept of search is familiar and widely used, it is not the only information extraction approach. When users have no previous knowledge of a dataset, or have not clearly defined their search task and are therefore unable to formulate a direct query, their task becomes one of *exploratory search* or *browsing* [150, 151] rather than *focused search* or *retrieval* [128]. Browsing approaches are more concerned with displaying all relevant information than with not showing any irrelevant information, which is the primary concern of traditional retrieval approaches [74, 103, 151].

More specifically, retrieval systems require the user to provide a search term (query) and seek to optimize the relevancy and ranking of the returned results. These systems do not factor in the entire information seeking process which often includes multiple queries and lengthy inspection of the query results [151]. Therefore, retrieval systems do not optimally support exploratory search tasks. In exploratory search “much of the search time in learning tasks is devoted to examining and comparing results” [151]. In the exploratory phase the user is actively involved in the search process and the results obtained are a combination of both the human’s do-

main knowledge and the information provided by the machine. Retrieval approaches are only concerned with the machine presenting the most relevant answer to the human, such as automatic question answering that is being built into search engines. Additionally, in their quest to provide the user with the most relevant answer, search engines are increasingly relying on personalization of search results, so that each user is more likely to receive the search results most relevant to them first, based on their previous search history. This creates a situation where users are always viewing an information source through their personal lens, which is referred to as a *filter bubble* [113] and where a user's personal views are always reflected back to them. Therefore, there is a need for tools that allow users unbiased access to a dataset and support users in exploring and querying unfamiliar datasets, in particular when they have not yet formulated a direct search goal.

Both search and browsing approaches are needed in different situations. However, while search and retrieval tools are widely provided, support for browsing of large datasets, especially rich semi-structured datasets which include complex relationships between data elements and are not simple to query, is lacking.

Semi-structured data is particularly difficult to explore and query because it does not always have a separate schema (i.e., it is self-describing), but in order to formulate SQL-like queries the user is required to have knowledge of the schema [43]. Treating semi-structured data as though it is complete free-text causes a loss of important additional meta-information which is encoded in the structured portions of the data. Treating semi-structured data by only considering the structured fields places less importance on the free-text portions which results in the loss of important information. In addition, rich semi-structured data is multi-faceted, can be multi-dimensional and documents can be related to each-other, making the data sources harder to explore without losing the relevant information contained in the additional dimensions.

In this dissertation we therefore develop a framework to support exploration of semi-structured data, which is otherwise difficult to gather insights from, without requiring the user to have prior knowledge of the dataset or have formulated a specific query. Our approach facilitates data exploration, analysis, and visualization by allowing the user to continuously update (i.e., add and remove) a set of keywords that the searched documents in the data set must contain. Our approach is novel in that the document set is not directly provided as the result of a specific query, but aggregated information and properties of relevant documents are provided. This reduces the time and mental effort spent examining and comparing each of the search results, and provides the user with further possibilities to continually refine the query. The presentation of aggregated information supports users in understanding

and learning about the dataset and enables them to formulate increasingly more accurate queries as their knowledge of the dataset increases and their search goal becomes clearer.

## 1.1 Problem Description

Large semi-structured datasets are not easily accessible for exploration as they cannot be effectively queried. Even existing approaches designed to support exploration of data still require the user to examine the result sets manually and aggregate the results mentally. In order to facilitate navigation in these datasets we need a uniform data structure. The data needs to be structured in a way that aids exploration and human understanding in order to provide optimal support for exploration tasks.

Semi-structured data is textual and we need a suitable visualization of this text that supports users in interacting with the data source and aids them in their query formation. In order to support a wide range of exploration tasks we also need a flexible navigation style that supports both refinement and broadening navigation steps interchangeably.

In order to provide a framework that efficiently supports exploration of a wide variety of semi-structured datasets, the process of extracting the relevant information from a large dataset and presenting it to users needs to be automated so that little or no manual pre-processing is required by the users. While there are already approaches catered to supporting a single task on a single dataset, these do not extend to a variety of semi-structured datasets in different domains. Therefore, we need a framework that is flexible enough to be applied in different domains and that supports the different tasks that arise in these domains.

## 1.2 Research Objectives

The main objective of this dissertation is to develop, implement and evaluate a generic framework through which large semi-structured datasets can be made available to the user in the context of exploratory search tasks, so that the burden of exploration and comparison of large result sets is removed from the user. We focus on the use of formal concept lattices to provide a suitable structure through which the data can be explored. We further investigate an alternative visualization for the data contained in our concept lattices as the lattice itself does not provide a suitable and scalable user interface. We investigate the use of tag clouds as a user interface through which navigation in an underlying concept lattice can be driven. We also investigate additional broadening navigation operations that further support the user in their exploration tasks. We apply our approach to three rich datasets, namely in-

dividual software repositories, collections of software repositories, which allow us to index additional information about the software projects themselves, and academic publication data, each with unique properties. We use our experiences with these datasets to drive further development of our approach.

In this dissertation we will:

- develop a framework for facilitating exploratory search in large semi-structured datasets;
- define methods for generating concept lattices from a variety of semi-structured data sources with different properties;
- develop broadening navigation in concept lattices which can be easily interchanged with traditional refinement navigation;
- develop tool support for automatically analyzing various semi-structured data sources and making these available for exploratory search; and
- apply our framework to three different data sources each with different properties.

Since a generic framework is not usable on its own, we instantiate our framework and apply it to different domains. We apply our framework to data contained in software repositories, in two different ways for different goals, to highlight the flexibility of the approach and the different tasks that can be supported using the same underlying dataset. We also instantiate our framework to support the exploration of a large collection of academic publication data. We evaluate each instantiation of our framework separately and perform case studies or user studies to evaluate how effectively users are able to use the tool or how the tool compares to the current available alternatives (e.g., GitK [13] and GitHub [8] for software repositories) to illustrate how our framework can be used to generate tools that provide functionality and flexibility that was previously unavailable in each domain. Each instantiation of our framework makes its own significant contribution to its individual domain (see Chapters 5 – 7 for further details). In addition, the application of our framework to different domains and tasks within those domains provides an evaluation for the flexibility and extensibility of the framework itself.

### 1.2.1 Visual Exploration of Software Development Repositories

Software version control archives such as Git and SVN contain a large amount of information. However, in trying to understand the software project the individual commits (which make up the documents in this context) are not relevant on their

own but form part of larger patterns which then provide interesting observations on the datasets. This dataset is therefore unlike other document collections as the documents here are representative of user's actions in a larger project as opposed to being self-contained. Manually investigating thousands of commits in a list-based presentation is an infeasible task for users and so the information contained in these repositories is not fully utilized in its potential to describe the development of the software project. Software repository data is also difficult to query because it is semi-structured with the most interesting information being provided in free text in the form of a comment on the current commit. Additionally, multiple data repositories (such as issue tracking and task tracking systems) are commonly used for a single software project (both of these features are provided on a single platform by the GitHub hosting site [8]) and so these repositories allow us to apply our approach to two different data-sources which are linked and are both relevant to understanding the software project and identifying patterns within the development process.

We apply our framework to software repository data in order to:

- support exploratory search on software development archives;
- evaluate how data from different sources (e.g., software development repositories and issue tracking systems) can be combined into the same context; and
- evaluate whether an exploratory search approach allows users to answer questions about their software repositories.

### 1.2.2 Identification of Skills from Aggregated Software Repository Data

Collections of Git repositories and users' commits within multiple projects provide a different dimension of information in the form of a user's skill set [89] which is evidence-based rather than self-authored (such as in the case of a CV). Analyzing collections of repositories allows patterns to be identified that may not be visible in a single repository. In the context of skills, individual commits may not provide much insight into a user's skill set, but when aggregated across all of a user's commits patterns can be identified which allow identification of a skill set. In the domain of hiring or skills evaluation the human is traditionally involved in the skills evaluation process [106, 145] and therefore the knowledge of the domain expert needs to be combined with the insights generated from the tool. Additionally, there are many other factors (such as seniority, location etc.) to be considered rather than just identifying the person with the highest level of skill in a particular area. Therefore, this dataset is well suited to an exploration approach rather than a retrieval or

recommendation-based approach as the domain expert is still involved in the process and able to identify and verify the most relevant information from the dataset.

We apply our framework to collections of software projects available on GitHub in order to:

- develop a tool that allows identification of relevant software developers from a large pool of developers in a specific location;
- evaluate how skills extracted from GitHub projects can be used in the recruitment process; and
- identify whether the current list-based interface provided by GitHub itself allows for accurate identification of software developers by asking developers about their experiences with recruitment on GitHub.

### 1.2.3 Exploration of Academic Publication Data

Academic publications provide an interesting application in that the publications are linked in the form of references and citations and these links themselves provide a large amount of information as to the quality and content of a particular dataset. Therefore, the dataset introduces the question of how these links between documents can not only be handled in the exploration process but exploited to build a more complete picture of the dataset. We have used the dataset of academic papers to conduct research on how different documents can be linked in an exploration approach and how navigation can be supported across document links.

We apply our framework to academic publication data in order to:

- support exploratory search of a large academic publication collection;
- demonstrate the scalability and flexibility of our approach;
- develop mechanisms for handling relationships between objects with concept lattices;
- develop a publication browser that presents aggregated citation and reference data; and
- evaluate the usability of our approach and the presence of learning effects for new users of the tool.



## 1.3 Outline of the Solution

According to White and Roth [151], exploratory search tools should support a number of functions including the “support of querying and rapid query refinement”, “offering facets and metadata-based result filtering”, “offer[ing] visualizations to support insight/decision making” and “support [for] learning and understanding”.

In order to support initial querying and consequent query refinements we make use of formal concept lattices as a navigation structure for our semi-structured datasets. Concept lattices facilitate step-wise navigation and are constructed in order to facilitate human understanding of datasets [153]. However, since large concept lattices cannot be suitably visualized [50] and concept lattices themselves require understanding in order to make them interpretable [69], large concept lattices do not provide a suitable interface for untrained users and are unsuitable to support users in gathering insight from large datasets. We make use of tag clouds to visualize the information at each navigation step in the underlying concept lattice as the semi-structured data is textual with the most relevant information often being contained in free text. The text visualization of the tag clouds emphasizes the most important aspect of the data and does not make the text a secondary aspect in the visualization as it would be if it were only serving the purpose of providing labels in a graph.

### 1.3.1 Research Methodology

In this dissertation we follow a “build and test” approach. We build a framework for exploration of semi-structured data and instantiate this for a variety of benchmark applications. We then compare the instances of this framework with customized tools previously developed for the same data analysis. We perform this comparison by conducting user studies with students, case studies in collaboration with industry partners and interviewing practitioners.

### 1.3.2 Technical Approach

In our approach we build a formal context from a given dataset, convert this context to a concept lattice and present the contents of the lattice in a suitable interface, which supports exploration of the underlying dataset.

#### 1.3.2.1 Formal Concept Lattice Construction

Concepts in a concept lattice are made up of objects and attributes. In our domain the objects are a set of documents (e.g., files in a Java project, revisions in a revision-control archive, projects in GitHub) and attributes are keywords and meta-data that we have extracted from the documents (e.g., methods in Java classes, words from

commit messages in a git repository, libraries used in a GitHub project). Concept lattices are constructed directly from a formal context table. Different context tables, and therefore different concept lattices, can be created from the same data archive by changing the document/object selection, e.g., in a git repository we can either take files or commits as objects. By changing the object selection and therefore constructing different context tables, our approach presents a different view on the same underlying data.

### 1.3.2.2 Interface Construction

Previous work on concept-based retrieval has made use of small customized list-based user interfaces [74,97] or focused on presenting a portion of the underlying concept lattices as an interface [81]. If we restrict the information presented in our interface such as in [81], then we also restrict the navigation paths available to the user, which is undesirable.

In order to provide an intuitive, scalable interface for exploration of semi-structured data we propose a tag cloud interface. Tag clouds are a simple and popular visualization method for textual data (often keywords) where the importance of each tag (typically its frequency in the document) is reflected in its size. We generate tags directly from the underlying data, instead of relying on user-generated labels for particular content, as commonly used in Web 2.0 applications (such as Flickr's [7] early tag cloud view). Tag clouds provide a visualization which places the most emphasis on the text.

The size of the tags in our tag cloud can be calculated from the number of documents in the dataset that contain the keyword that the tag represents. In Information Retrieval (IR) [102] terms, our tag cloud can be seen as the aggregation of the Boolean term frequencies for each document in the query result, scaled according to the size of the document collection. The concept lattice then provides an efficient way to compute our tag cloud; a computation from only the inverted index would be impractically inefficient: we would first need to retrieve all documents indexed by the selected tags, then iterate over the entire vocabulary and compute the size of the intersection of each term's inverted index with the query's result. Hence, any efficient IR-based implementation must use the same information in essentially the same way as a lattice-based implementation. However, we can exploit the lattice's support for browsing [74].

Our approach builds on concept-based retrieval and browsing. However, unlike previous work [74,97], our tag cloud interface displays the implied ranking of the keywords in the retrieval set by using the tag size to indicate to how many documents the keyword applies. Our tag cloud can be seen as a visual representation of

unexplored queries, highlighting the most prominent navigation path. We display all keywords that occur at least once in any document that matches our query (the selected tags) and calculate the size/importance of their tags according to how often they occur. Tags are scaled according to the total number of documents in the dataset so that tag sizes continue to reflect the occurrence of the keyword over the full document set when the set of documents matching the query becomes smaller.

### 1.3.2.3 Support for Navigation

We primarily make use of a refinement navigation approach, allowing users to incrementally select tags that make up the current query. However, since tags can be de-selected in any order, our approach also indirectly supports pivot navigation [109], for example if a user wishes to pivot from tag A to tag B they can initially select tags A and B, and then de-select tag A to have only tag B selected. In this way, users are mindful of their navigation path from one tag to another, having first seen the results for both items selected and observing what attributes or objects they have in common before pivoting.

Refinement navigation in the concept lattice is step-wise and the focus concept is updated at each navigation step [50, 97]. Our tag clouds are generated directly from the focus concept in the lattice and are updated on each navigation step. If the selected set of keywords is interpreted as the query then the extent (i.e., set of objects or, in our case, documents) of the focus concept can be seen as the query's result. Our navigation is driven by updates (i.e., additions and deletions) to a set of keywords that the searched documents in the dataset must contain.

All tags available for refinement selection in our tag cloud will either further refine the document set or leave it unchanged. As in previous work [74, 97] the user is prevented from constructing a query that returns an empty set of documents. By de-selecting tags in a different order in which they were selected the user is able to navigate out using a different navigation path, which indirectly supports pivot navigation and facilitates exploration of the dataset.

The standard broadening operation in concept lattices uses the join in the lattice. However, for structured fields the join operation can overgeneralize and, hence move the focus to the top of the lattice, effectively resetting the navigation and removing all previous selections. For example, with a version control archive with revisions as documents the join on multiple year keywords would move the focus to the top of the lattice since no revision can have two commit dates. We develop a broadening navigation algorithm that performs as a Boolean or-operator (e.g., returning revisions in 2002 *or* 2004) instead of using the join in the lattice, as well as investigating other operations that support selections for generalization.

### 1.3.3 ConceptCloud Browser

We have constructed the ConceptCloud browser (available at [www.conceptcloud.org](http://www.conceptcloud.org)), which creates interactive tag clouds. The browser can load XML and JSON data and allows the user to choose a field of the data that will be used as the document/object (e.g., file in version control repository) and the fields that should be used as keywords for the document (e.g., authors that have changed the file). Our browser also supports various pre-processing options such as splitting large text fields, merging fields and stemming. We color tags in the interface according to the category of information that they represent (such as commit author, file etc.) and different categories can be displayed in linked tag cloud views to provide a multi-faceted view. Selected tags (forming the current query) are displayed in red in order to distinguish them. We use the incremental approach of [85] in order to construct concept lattices in our prototype. Concepts are computed on the fly eliminating the need to construct the entire lattice. This mitigates the large initial indexing times usually associated with concept lattices.

While the browser supports the exploration of any dataset that can be input in the correct format, we have also instantiated three specific instances of the browser. Our version control browser can automatically analyze Git and SVN repositories, and can combine their information with that extracted from an issue tracking system. Our skills browser can take as input a specific location e.g., "Cape Town", perform a lengthy off-line indexing phase, and return a tag cloud of skills for developers in that location, extracted from their GitHub profiles. Our publication browser can take as input a large JSON file of academic publications and present these in an interactive tag cloud that also supports exploration of reference and citation data.

## 1.4 Contributions

The main contribution of this PhD dissertation is the development of a framework that supports exploration tasks on a variety of semi-structured data sources and the application and evaluation of this framework in three different domains. In the construction of a generic framework this dissertation makes novel contributions which include an approach for constructing an interactive tag cloud from data contained in a formal concept lattice, a broadening navigation approach in concept lattices and the development of the framework itself.

We describe how large collections of semi-structured data can be used to construct a formal concept lattice which forms a structure for navigation. We describe a step-wise navigation algorithm that supports both refinement and broadening navigation algorithms interchangeably and indirectly supports pivot navigation as well. We

show how a tag cloud can be generated from a focus concept in the concept lattice and how navigation in the concept lattice can be driven by selections and de-selections in the tag cloud.

We instantiate our framework in three different domains and make novel contributions in each of these domains. We have built and described a flexible software repository browser that enables exploration of software repositories and is flexible enough to support a variety of tasks. Our ConceptCloud Version Control Archive browser can take as input a Git or SVN repository and present this to the user in an interactive tag cloud which allows them to answer questions about the development process which have been identified in [52, 132].

We have instantiated our framework in the form of a GitHub skills browser which has been applied and evaluated in the software developer recruitment domain. Our skills browser allows for the identification of candidate developers from a large pool of developers and allows candidates to be identified based on their exhibited skills and not just subjective skill indications that can be gathered from their profiles in a limited amount of time.

We have used our framework to build a flexible academic publication browser which can be used to show aggregated publication data as well as reference and citation data. Our browser allows users to answer complex questions about author activities, citation analysis and author collaborations. We have evaluated our publication browser in the form of a user study. The application of our framework to a large dataset of academic publications also demonstrates the scalability and flexibility of our approach.

## 1.5 Organization of the Dissertation

This introduction has provided a context for our approach and provided details as to our technical approach and contributions made in this research.

In Chapter 2 we further provide background information for the use of formal concept lattices as well as for exploratory search approaches and tag clouds. We detail the mathematical foundations on which our approach is built and show how formal concept lattices have been used to support navigation. We describe the fundamental aspects of exploratory search and the features that are needed by tools in order to support exploration of data. We also discuss tag clouds as a visualization for textual information.

We then describe our refinement and broadening navigation approaches in concept lattices in Chapter 3. Our refinement navigation approach allows users to continually refine their result sets. We provide a de-selection which indirectly sup-

ports pivot navigation by allowing users to de-select items in any order (not just as an undo operation). We also describe various broadening approaches which can be applied interchangeably with refinement operations.

We then describe the implementation of our approach in the ConceptCloud browser which provides a framework that can be customized for a variety of different applications in Chapter 4. We also show how a tag cloud can be built directly from a concept in the concept lattice. ConceptCloud provides a web application framework which can be instantiated in different domains.

We describe how our approach can be applied to version control repositories and bug archives in Chapter 5. We also provide an evaluation for our repository browser in the form of case studies on open source and industrial projects, and conduct a user study.

We then show how our browser can be used to handle data from multiple repositories and supports skills browsing of developers on GitHub in Chapter 6. We survey developers and recruiters to gain insight into the use of GitHub for recruitment tasks and establish the limiting features of GitHub profiles when used for recruitment. We then build a skills browser using our generic framework and evaluate the browser via feedback from recruiters and using the browser to recommend developers for open positions.

We use our approach to construct a publication browser for data contained in a digital library dataset (specifically the ACM Digital Library [34]) in Chapter 7 and evaluate the publication browser by conducting a user study. We also show how object relationships (citations and references in this domain) can be handled with concept lattices and demonstrate the scalability of our approach on a dataset containing 1.5+ million objects.

We draw conclusions and discuss directions for further research in Chapter 8.

## **Declaration of Joint Work**

Parts of this dissertation have been done in collaboration with others. Marvin Esterhuizen has implemented the ConSL scripting language (see Section 4.4) that plugs into ConceptCloud and allows users to script different combinations of viewers. Further details of his implementation are available in [71]. Jean Breytenbach has implemented the initial ConceptConstructor add on (see Section 4.2.1) for ConceptCloud which facilitates the reading of generic XML and JSON files. Chapter 7 is partly joint work with Marcel Dunaiski. Marcel has performed the key-phrase extraction for the academic publication data (see [66] for further details). The user study in Chapter 7 was also done in collaboration with Marcel Dunaiski where we jointly set up the question set, observed the participants and analyzed the results.

With regards to the publication list presented on page vi, paper 6 was joint work with Marcel Dunaiski. Marcel has developed and implemented the key-phrase extraction technique, and I have adapted and applied the ConceptCloud browser to the publication data. We have also jointly conducted the user study, where we jointly set up the question set, observed the participants and analyzed the results. Paper 7 was joint work with Marvin Esterhuizen. Marvin has implemented the ConSL scripting language. Bernd Fischer has supervised all conference and journal papers.

## Chapter 2

# Background

In this chapter we describe the foundational work for our approach. This dissertation builds on research in the areas of formal concept analysis, particularly navigation in concept lattices, and approaches for visualizing the data contained in a concept lattice, and exploratory search, particularly navigation styles and visualizations employed in data exploration approaches. We also make use of a tag cloud visualization which is described in this chapter.

### 2.1 Formal Concept Analysis

We describe only the basic notations for Formal Concept Analysis and provide a small illustrative example of a context table and concept lattice. We refer to the cited literature for further details.

#### 2.1.1 Mathematical Foundations

Formal Concept Analysis (FCA) [59, 77, 152] uses lattice-theoretic methods to investigate abstract relations between objects and their attributes. Such *contexts* can be imagined as cross tables where the rows are objects and the columns are attributes.

**Definition 1** *A formal context is a triple  $(\mathcal{O}, \mathcal{A}, \mathcal{I})$  where  $\mathcal{O}$  and  $\mathcal{A}$  are sets of objects and attributes, respectively, and  $\mathcal{I} \subseteq \mathcal{O} \times \mathcal{A}$  is an arbitrary incidence relation.*

Figure 2.1 shows an example of a small context table. The table contains data from wine reviews of four bottles of wine. The columns are used to describe the attributes (or properties) of the bottles of wine. The rows are used to describe the objects which are individual bottles of wine in this example. Whenever an attribute applies to a particular object (bottle of wine) a cross is placed in the context table to indicate the relationship.



	{country}		{review text}		{varietal}		
	USA	South Africa	Fruity	Berry	Cabernet Sauvignon	Merlot	Pinotage
wine-bottle1	X		X		X		
wine-bottle2		X	X	X	X		
wine-bottle3	X			X		X	
wine-bottle4		X		X			X

Figure 2.1: Example context derived from wine review data. The bottles of wine are objects with review text, country and varietal as attributes. Attribute facets are indicated by the color of the column.

**Definition 2** Let  $(\mathcal{O}, \mathcal{A}, \mathcal{I})$  be a context,  $O \subseteq \mathcal{O}$ , and  $A \subseteq \mathcal{A}$ . The common attributes of  $O$  are defined by  $\alpha(O) = \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in \mathcal{I}\}$ , the common objects of  $A$  by  $\omega(A) = \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in \mathcal{I}\}$ .

Note that for  $o \in O$  we denote  $\omega(\{o\})$  by  $\omega(o)$ .

Concepts are pairs of objects and attributes which are synonymous. They are maximal rectangles (modulo permutation of rows and columns) in the context table.

**Definition 3** Let  $(\mathcal{O}, \mathcal{A}, \mathcal{I})$  be a context. Then  $c = (O, A)$  is called a concept of  $\mathcal{C}$  iff  $\alpha(O) = A$  and  $\omega(A) = O$ . The extent and intent of  $c$  are given by  $\pi_O(c) = O$  and  $\pi_A(c) = A$ , respectively. The set of all concepts of  $\mathcal{C}$  is denoted by  $B(\mathcal{C})$ .

Concepts are partially ordered by inclusion of extents such that a concept's extent includes the extent of all of its subconcepts; the intent-part follows by duality.

**Definition 4** Let  $(\mathcal{O}, \mathcal{A}, \mathcal{I})$  be a context,  $c_1 = (O_1, A_1), c_2 = (O_2, A_2) \in B(\mathcal{C})$ .  $c_1$  and  $c_2$  are ordered by the subconcept relation,  $c_1 \leq c_2$ , iff  $O_1 \subseteq O_2$ . The structure of  $B(\mathcal{C})$  and  $\leq$  is denoted by  $\mathcal{B}(\mathcal{C})$ .

The basic theorem of FCA states that the structure induced by the concepts of a formal context and their ordering is always a complete lattice. Such *concept lattices* have strong mathematical properties and reveal hidden structural and hierarchical properties of the original relation. They can be computed automatically from any given relation between objects and attributes. The greatest lower bound or *meet* and least upper bound or *join* can also be expressed by the common attributes and objects.

**Theorem 5 (Wille, [152])** Let  $\mathcal{C}$  be a context. Then  $\mathcal{B}(\mathcal{C})$  is a complete lattice, the concept lattice of  $\mathcal{C}$ . Its meet and join operation for any set  $I \subset B(\mathcal{C})$  of concepts

are given by

$$\bigwedge_{i \in I} (O_i, A_i) = \left( \bigcap_{i \in I} O_i, \alpha(\omega(\bigcup_{i \in I} A_i)) \right), \text{ and}$$

$$\bigvee_{i \in I} (O_i, A_i) = \left( \omega(\alpha(\bigcup_{i \in I} O_i)), \bigcap_{i \in I} A_i \right)$$

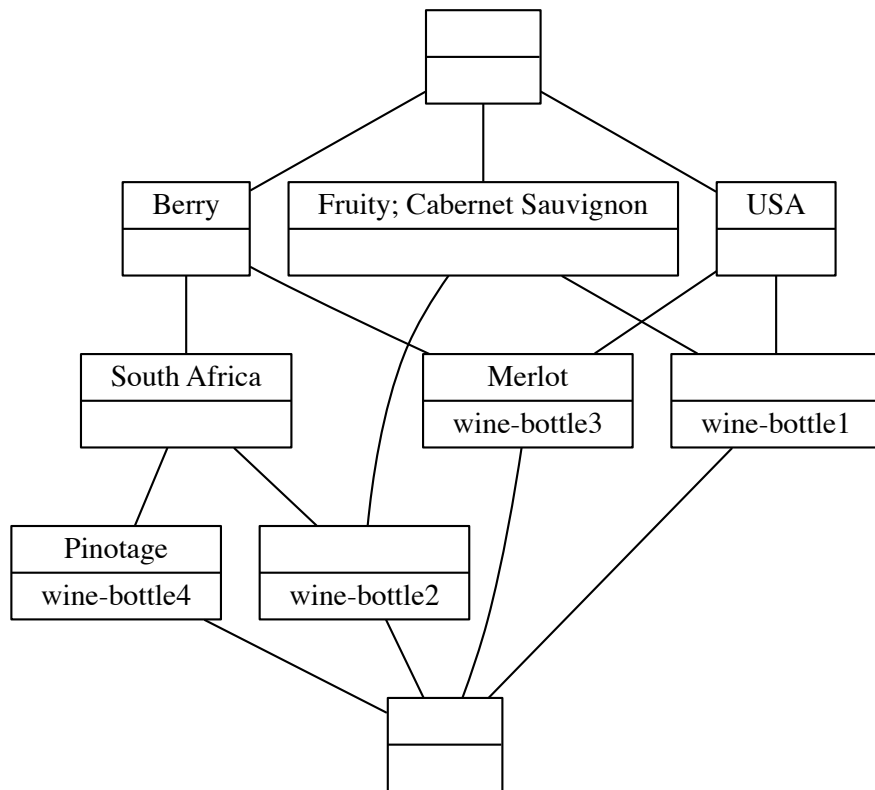


Figure 2.2: Example of a concept lattice derived from the context in Figure 2.1. Concepts are linked according to their relationships with each other: a downwards link indicates that the lower concept inherits all attributes from the linked higher concepts; an upwards link indicates that the concept inherits all objects from the linked lower concepts. Object and attribute labels are not repeated. An attribute label is provided at the highest concept to which the attribute applies and an object label is provided at the lowest concept to which the object applies.

Figure 2.2 shows an example of a small concept lattice generated from the data in the context table provided in Figure 2.1. The top concept in the lattice contains all objects (wine bottles) in the dataset and only attributes that apply to every object in the dataset (in this case no attributes). The bottom concept in the lattice contains all attributes in the context and only objects for which every single attribute applies (in this case none). Towards the bottom of the lattice the concepts have less objects

and more attributes and provide more specific descriptions of the objects. In the lattice there exists a concept which contains the full attribute set for each object, which is referred to as the object concept or introducing concept of the particular object. Note that the extent of this concept might contain more than one object if there are two objects in the context table which have exactly the same attributes. Not all object concepts are at the atom level of the lattice as there may be an object that has an attribute set which is a subset of another object's attribute set. Higher up in the lattice there is also a concept which contains all objects for which a specific attribute applies, which is referred to as the attribute concept or introducing concept of the particular attribute. Each concept in the lattice contains a set of attributes (the concept's intent) and a set of objects (the concept's extent) which are synonymous with each other.

**Definition 6** *Let  $\mathcal{B}((\mathcal{O}, \mathcal{A}, \mathcal{I}))$  be a concept lattice. The defining concept of an attribute  $a \in \mathcal{A}$  (object  $o \in \mathcal{O}$ ) is the greatest (smallest) concept  $c$  such that  $a \in \pi_{\mathcal{A}}(c)$  ( $o \in \pi_{\mathcal{O}}(c)$ ) holds. It is denoted by  $\mu(a)$  ( $\sigma(o)$ ). We use  $\delta(x)$  to denote  $\mu(x)$  if  $x$  is an attribute and  $\sigma(x)$  otherwise.*

Efficient algorithms exist for the computation of the concept lattices and the meet and join of concepts in the lattice [96].

### 2.1.2 Visual Presentation of Data Contained in Concept Lattices

Concept lattices themselves are traditionally visualized as Hasse diagrams [152]. However, for large datasets, the Hasse diagrams become incomprehensible and no longer provide a suitable visualization of the lattices. Figure 2.3 shows an image of a large concept lattice generated from the structure of an aerodynamics system in [99]. It is obvious that the lattice is too large to enable users to interpret the relationships between the concepts and derive any information from the lattice structure.

Alternative approaches for presenting the information contained in concept lattices have made use of small customized list-based user interfaces [74, 97] or focused on presenting a portion (usually the neighboring concepts of the current focus) of the underlying concept lattices as an interface [81]. Carpineto and Romano have also made use of a fish-eye view for presenting portions of the lattice [48]. A representation of the area beneath the focus as a tree has also been used as a visualization [54].

The LatViz tool supports interactive visualization in concept lattices directly and also supports filtering of the lattices themselves [37].

The Credo system [50] uses a list-based interface that displays the sub-concepts of the current focus but also indicates the number of times that a particular document appears in the sub-concepts. Items in the list can then be selected to generate a new



gle query, and not step-wise navigation through the concept lattice, which follows a process of incremental refinement or query formation. However, there has been some research on exploring different step-wise navigation techniques in concept lattices. Some navigation techniques allow small step-wise increments of navigation to neighboring concepts in the lattice [82] while other approaches make use of larger-step navigation algorithms which are more suitable for large data collections [74, 97].

Godin et al. [82] described an iterative retrieval algorithm which maintains a *focus* concept whose extent is the retrieval result. Initially, the focus is the lattice's top element; in each iteration the user moves the focus to an adjacent concept, by adding or removing an attribute in the intent (or not in the intent) of a concept directly above or below the current focus. However, this navigation style is too incremental, because the focus can move only one level at a time, and too constrained, because the user can only choose attributes from the intents of the directly adjacent concepts, and has no indication which choices are hidden behind paths not taken.

Lindig [97] introduced a semi-constrained navigation algorithm where the focus can be refined by selecting *any* attribute from *any* concept (except  $\perp$ ) below the focus, provided the attribute is not already in the focus' intent. The focus is then updated by computing its meet with the attribute concept as shown in Theorem 8. A restriction on selectable attributes (which Lindig calls "significant keywords") shown in Definition 9 prevents navigation into dead ends, and ensures that each query refinement also refines the query results. The meet operation is widely accepted as a suitable method for refinement navigation and supports conjunctive queries.

Following Lindig [97] a query (which we refer to as a navigation step) and the query result (which we refer to as the extent of the focus concept) can be defined as below.

**Definition 7** [97] *A set  $A \subseteq \mathcal{A}$  is a query to a concept lattice  $\mathcal{B}((\mathcal{O}, \mathcal{A}, \mathcal{I}))$ . A component  $o \in \mathcal{O}$  satisfies a query iff  $\omega(o) \supseteq A$  holds. The set of all components satisfying a query is called a result and is denoted by  $[[A]] \stackrel{\text{def}}{=} \{o \mid o \in \mathcal{O}, \omega(o) \supseteq A\}$ .*

**Theorem 8** [97] *Let  $\mathcal{B}((\mathcal{O}, \mathcal{A}, \mathcal{I}))$  be a concept lattice and let  $A \subseteq \mathcal{A}$  be a query. Then  $[[A]] = \pi_{\mathcal{O}}(\bigwedge_{a \in A} \mu(a))$  holds.*

**Definition 9** *Let  $A \subseteq \mathcal{A}$  be a query to  $\mathcal{B}((\mathcal{O}, \mathcal{A}, \mathcal{I}))$ . The set of significant keywords is denoted by  $\langle\langle A \rangle\rangle \stackrel{\text{def}}{=} \{a \in \mathcal{A} \mid \emptyset \subset [[A \cup \{a\}]] \subset [[A]]\}$ .*

Fischer [74] exploited the duality of concept lattices and introduced object-based navigation; here, selection of an object not in the focus' extent is a widening step

that is implemented via the join. This provides a broadening operation to move the focus higher in the lattice.

Note that while there has also been some further work on other broadening approaches, such as disjunctive navigation, this is discussed in Chapter 3 to make the comparison with our own navigation style clearer.

## 2.2 Exploratory Search

Exploratory search (or browsing) approaches support data exploration even when the user has not yet formed an explicit query or is unfamiliar with the dataset. Exploratory search tasks cover learning and investigating the data set in order to make observations as opposed to only direct lookups. Users engaging in exploratory tasks often need to learn about the dataset in order to understand how their search goal can be achieved [151].

The field of exploratory search is a sub-domain of information seeking [95] which is different from information retrieval [102] in that information retrieval assumes that the relevant information exists in a data source and can be extracted with a properly formed query, whereas in information seeking the user does not know whether an answer to their query exists or not.

Marchioni [103] defines exploratory search in the following way:

“Exploratory search can be used to describe an information seeking problem context that is open-ended, persistent and multi-faceted; and to describe information-seeking processes that are opportunistic, iterative and multi-tactical. In the first sense, exploratory search is commonly used in scientific discovery, learning, and decision-making contexts. In the second sense, exploratory tactics are used in all manner of information seeking and reflect seeker preferences and experience as much as the goal.”

White and Roth note that exploratory search is “as much about the journey through the information space as the destination” [151] because as users explore the dataset they learn more about the context and are able to formulate increasingly more accurate search queries.

Exploratory tasks typically involve multiple iterations and results are not immediately expected on the first query attempt, as they are with retrieval approaches. Exploratory search approaches heavily depend on human-interaction and therefore insights that are gathered during exploration also comprise the domain knowledge of the human interpreting the information presented. During exploration, users may notice interesting patterns in the data that they may have been unlikely to query for without having had the opportunity to observe the dataset.

Interfaces supporting exploratory search can allow the user to explore the full dataset and be cognizant of where a particular document lies in comparison to the other documents in the result set, which is not possible when users examine the results one document at a time when they are presented in a ranked list format [151].

According to White and Roth [151], exploratory search tools should support a number of functions including the “support of querying and rapid query refinement”, “offering facets and metadata-based result filtering”, “offer[ing] visualizations to support insight/decision making” and “support [for] learning and understanding”. Exploratory search tools therefore aid in highlighting relevant and interesting navigation paths for the user and allowing them to follow these paths.

### 2.2.1 Navigation Styles for Exploration of Datasets

Various navigation processes have been employed in data exploration approaches [64, 87, 151, 154]. Approaches can be largely classified into either a refinement navigation style or a pivoting navigation style, where the pivoting style of navigation does not make incremental refinements on the current result set but rather changes the entire result set when a new selection is made. With a pivoting navigation style on each new selection, the result set is completely changed or shifted [109]. The pivoting navigation style can be thought of as supporting queries which are always applied to the full dataset (always clearing the previous selection history), whereas the refinement navigation style applies each new query to the result set of the previous queries.

Examples of approaches making use of pivot navigation styles are provided in [64, 87] while [134, 154] provide examples of a refinement-based navigation style.

Gwizdka and Bakelaar’s work on Tag Trails [87] uses a tag cloud to facilitate navigation in the pivot style. They also make use of the concept of ‘breadcrumbs’ which preserves the navigation history by presenting the tags for historical selections and displaying the previous tag clouds so that the user is aware of how their navigation path has led them to the current result set.

The pivot paths approach used in [64] uses a pivoting navigation approach but seeks to make the changes between different subsets of results more gradual. The approach uses pivoting but also includes a comparison view which allows two selections to be compared to each other. The previous selection can always be displayed alongside the new selection to provide some history to the pivoting process.

Yee et al. [154] use a refinement-based navigation approach in order to facilitate exploration of image collections. The query process can be started either by searching for a specific term or selecting a term from the available options. Once an initial query has been made, additional terms are then available for refinement. Once a

desirable document (image in this application) has been found it can be displayed along with its searched query terms.

The Magnet tool [134] supports exploratory search on semi-structured data archives. While there are a variety of navigation modes provided by the tool in “navigation advisors”, one of these options is refinement-based. The tool is applied to a collection of recipes and the interface allows users to see lists of recipes for which selected properties (such as ingredients or cooking methods) apply to. For the refinement style navigation additional properties of recipes that apply to some but not all of the recipes shown at a specific navigation step can be selected to refine the recipe set. Selections can be made on a pane to the left of the recipe list and for each available selection the number of recipes that apply to the property is provided in brackets next to the name of the property.

### 2.2.2 Visual Presentation of Data for Exploration

There are multiple approaches for presenting data in formats that specifically facilitate exploration. Data available for exploration can be presented along different facets (categories of information) [119], allowing the user to make refinements in a particular facet or in multiple facets.

Some approaches have made use of tags to facilitate navigation [87, 154]. For example, Figure 2.4 shows the Tag Trails interface from [87]. The interface shows tag clouds for previous navigation steps in order to provide a context for the current navigation step, since the navigation uses a pivot style which does not preserve history.

Another approach has made use of a more scatter-plot-like visualization, shown in Figure 2.5, that allows dynamic filtering based on various fields [36]. The scatter plot will update whenever any refinements are made on any of the facets provided.

Timeline visualizations have also been used to facilitate data exploration as shown in [124]. A timeline visualization from [124] is shown in Figure 2.6.

Interactive graph-based visualizations, such as the one employed in Elastic Search’s Kibana Visualization tool [18], can also support navigation by allowing users to select graph segments. All graphs can then be linked so that when one item is selected all other graphs update to indicate how the selection affects all facets of information.

A common theme in most exploration systems is that they initially show a summary of the underlying data and allow the user to pivot or make refinements by selecting a feature of the summary provided, which aids the user by providing suggestions for the start of their navigation path. This style of interface design (which supports dynamic querying) is highlighted by White and Roth [151] as a requirement for exploratory search tools.



**Tag Trails**  
Results for tag phylogeny. 701 articles.

New Taxonomy and the Origin of Species  
*PLoS Biology*, Vol. 5, No. 7, (1 July 2007), e194.  
by Shai Meiri, Georgina M Mace  
tags [evolution](#) [phylogeny](#) [specy](#)

A Phylogenetic Method for Detecting Positive Epistasis in Gene Sequences and Its Application to RNA Virus Evolution  
*Mol Biol Evol*, Vol. 23, No. 9, (1 September 2006), pp. 1724-1730.  
by Beth Shapiro, Andrew Rambaut, Oliver G Pybus, Edward C Holmes  
tags [evolution](#) [phylogeny](#) [positiveepistasis](#)

Measures of Clade Confidence Do Not Correlate with Accuracy of Phylogenetic Trees  
*PLoS Computational Biology*, Vol. 3, No. 3, (1 March 2007), e51.  
by Barry G Hall, Stephen J Salipante  
tags [accuracy](#) [clade](#) [confidence](#) [phylogeny](#)

Predicting Protein Function with Hierarchical Phylogenetic Profiles: The Gene3D Phylo-Tuner Method Applied to Eukaryotic Genomes  
*PLoS Computational Biology*, Vol. 3, No. 11, (1 November 2007), e237.  
by Juan A Ranea, Corni Yeats, Alastair Grant, Christine A Orengo  
tags [eukaryota](#) [genome](#) [hierarchical](#) [phylogeny](#) [prediction](#) [protein](#)

Evolution of genes and genomes on the Drosophila phylogeny  
*Nature*, Vol. 450, No. 7167, (November 2007), pp. 203-218.  
tags [eukaryota](#) [genome](#) [hierarchical](#) [phylogeny](#)

[Phylogenetic Inference Using Whole Genomes](#)  
*Annual Review of Genomics and Human Genetics*, Vol. 9, No. 1, (2008), pp. 217-231.  
by Bruce Rannala, Ziheng Yang  
tags [phylogeny](#) [genome](#)

Toward Resolving the Eukaryotic Tree: The Phylogenetic Positions of Jakobids and Cercozoans  
*Current Biology*, Vol. 17, No. 16, (21 August 2007), pp. 1420-1425.  
by Naiara Rodriguez-Espeleta, Henner Brinkmann, Gertraud Burger, Andrew J Roger, Michael W Gray, Hervé Philippe, Franz B Leng  
tags [eukaryota](#) [evolution](#) [phylogeny](#) [specy](#)

High-resolution species trees without concatenation  
*PNAS*, Vol. 104, No. 14, (3 April 2007), pp. 5936-5941.  
by Scott V Edwards, Liang Liu, Dennis K Pearl  
tags [phylogeny](#) [specy](#)

[Change settings](#). Click here to change settings for how the tag clouds are displayed.

**Current: phylogeny**

**algorithms** alignment analysis animals characters classification dna evolution genetic genetics genome likelihood mammals methods mitochondrial models molecular nuclear parsimony phyloinformatics primates protein research sequence software source-trees speciation support taxonomy topology vertebrates

Total: 1629

**1 ago: algorithms**

alignment analysis biological comparative computational computer data databases dna gene genes genetic genome govt graph humans linkage methods models molecular networks non-phs phs phylogeny protein proteins refmanager reproducibility research results sequence simulation software statistical structure support

Diff: 2220 Total: 2997 tags

**2 ago: retrieval**

algorithms analysis context database databases factual gpubmed humans image indexing information intelligence management methods small storage systems topic

Diff: 714 Total: 935 tags

**3 ago: information**

data databases health human humans internet library management methods model models recognition research retrieval sequence small statistics storage support systems technology theory time topic user visualization web

Diff: 2806 Total: 3168 tags

Figure 2.4: The Tag Trails interface as shown in [87] which uses a pivot navigation style and shows a history of navigation steps.

**Film Finder**

Scatter plot showing movie data points (years 1920-1995 on x-axis, ratings 0-9 on y-axis). Points are color-coded by genre: Drama (red), Mystery (blue), Comedy (yellow), Music (green), Action (orange), War (purple), SF (brown), Western (pink), Horror (black).

**Filters:**

- Title: ALL
- Actor: ALL
- Actress: ALL
- Director: ALL
- Length: 0 to 450
- Ratings: G, PG, PG-13, R

Copyright (C) 1993 HCL

Figure 2.5: The Film Finder interface as shown in [36] which uses a scatter plot interface that supports dynamic filtering.

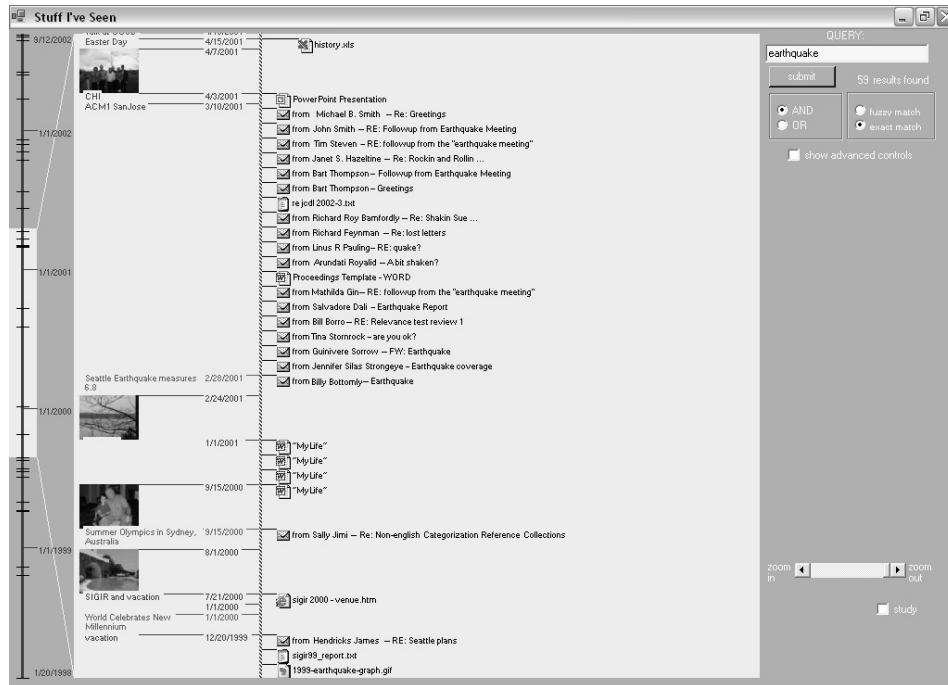


Figure 2.6: The Stuff I've Seen interface which uses a timeline visualization as described in [124].

## 2.3 Tag Clouds

Tag clouds (or word clouds) are a simple visualization method for textual data where the frequency of each tag is reflected in its size. Tag clouds are typically generated on social sites where content is “tagged” by users in order to indicate the topic of the content [88]. Therefore tag clouds have evolved as a mechanism to navigate directly to content with a particular “tag” [88].

The simplest and most popular tag cloud layout [101] is as an alphabetically sorted list of tags in a roughly rectangular shape which was found by Schrammel et al. to perform better than random or semantic layouts [130]. A variety of alternative tag layout methods have been proposed, such as tag flakes by Caro et al. [46]. Tag flakes are used in order to provide context for tags as basic tag clouds fail to show how the tags are related [46].

Tag clouds are traditionally static text visualizations, which provide a summary for a piece of text or tags that have been generated from a tagging system. For example, a well-known example of a tag cloud is one constructed from Obama and Bush’s state of the nation addresses shown in Figure 2.7.

However, there has been some work on supporting navigation in tag clouds. Navigation using tag clouds has previously been explored using a Bayesian approach [108]; however, navigation in our browser is supported by a novel combination of tag

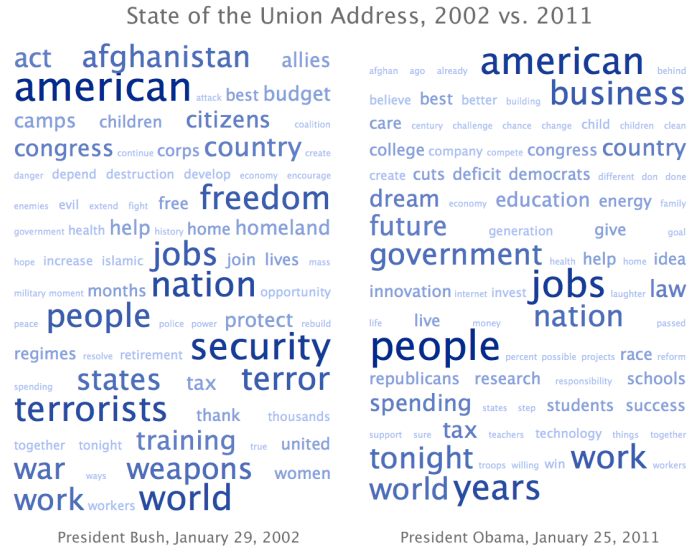


Figure 2.7: Example of a tag cloud: Comparison of text from President Obama’s and President Bush’s state of the nation addresses [31]. The importance (here frequency) of each word is indicated by its size.

clouds and concept lattices [59, 77, 152].

In web applications tags in the tag cloud have also been used as links which would navigate to the portion of documents that have been “tagged” with the selected tag. This can be seen as a pivot navigation style, which supports a top-level query on the full document set.

## Chapter 3

# Single-Focus Navigation in Concept Lattices

In this chapter we describe our single-focus refinement and broadening operations in concept lattices. Our navigation style maintains a single focus on all navigation steps so that different types of navigation (i.e., refinement and various styles of broadening navigation) can be used interchangeably. Refinement in concept lattices is a well-defined operation which makes use of the meet operation in the lattice. We might thus expect that the join operation (which is the inverse of the meet operation) would support generalization or broadening approaches, but this does not yield intuitive results on large datasets. We describe our refinement selection and de-selection approaches (see Section 3.2) which support refinement navigation and indirectly support pivot navigation. We also describe various broadening navigation operations, such as a boolean OR operation (see Section 3.4), a generalization operation (see Section 3.5) which moves the focus concept up in the concept lattice and a “more like this” operation (see Section 3.6) which generalizes from one object to other similar objects.

### 3.1 Introduction

Concept lattices can in principle be navigated directly, by following the subconcept relation to move from one concept to another concept in its direct neighborhood [81]. However, this only allows for small navigation steps and thus restricts the serendipitous nature of the browsing operation and becomes impractical for large lattices. Instead, in our approach we aim at large-step navigation algorithms that allow users to select and deselect arbitrary attributes and rely on the meet and join operations to move between concepts [97].

Large-step navigation algorithms should ideally satisfy a number of properties

that ensure that their behavior is transparent to users. First, they should be *functional*, i.e., rely only on the current query concept and the new selection (or de-selection) to determine the next concept as result of the navigation step. This means that users do not need to remember the navigation history in order to understand the results. Second, they should be *commutative*, i.e., the order of the navigation steps should have no effect on the next navigation result. This allows users a certain degree of freedom in how they navigate through the underlying document collection. Finally, they should have the *single-focus* property, i.e., each query result can be represented by a single concept in the lattice. The single-focus property enables different styles of navigation to be used interchangeably so that we can apply various broadening navigation operations after a refinement selection and vice versa. The functional and commutative properties then ensure that when navigation styles are used interchangeably the results are still intuitive.

If we follow a purely conjunctive query interpretation (i.e., consider all query terms to be connected by the AND operator), we can use the lattice's meet operation as implementation of the AND operator [49] in a document-term concept lattice. Moreover, the navigation algorithm is then by construction functional and commutative, and has the single focus property.

We might expect that the join operation would support intuitive broadening navigation. However, when navigating in large datasets the join of two concepts with large extents calculates the intersection of the attribute sets for each concept which can easily become empty. The focus then moves back to the top of the lattice, effectively resetting the navigation and removing all previous navigation steps. Therefore, we develop new operations to support generalization from both an additional attribute selection and from a single object. We also develop a mechanism for supporting disjunctive queries with a single focus in the underlying lattice.

Additionally, in semi-structured data it can be useful to consider the different facets [118] in the construction of the contexts. For example, when USA appears as the country of origin in a wine dataset this provides different information to when USA simply appears in the wine review text. For this reason in our approach we would consider USA as country of origin and USA as wine review text as different attributes in the context. Some facets can also be functional (such as the vintage of a wine bottle) where only one attribute from the particular facet can apply to an object.

The consideration of facets further complicates the navigation in the lattices, as the selection of two attributes from a functional facet for refinement would always lead to an empty result set, since by definition no object can have more than one attribute from a functional facet. For the broadening navigation operation the pres-

ence of two attributes from a functional facet would result in the attributes canceling each other out and therefore no attributes from that facet would be present in the concept resulting from the join operation, which is counter-intuitive.

## 3.2 Refinement Navigation

### 3.2.1 Refinement Selection

Refinement operations in the lattice can be implemented using the meet operation. For step-wise navigation, we maintain a current focus concept at each navigation step. The focus can be refined with a new selection by calculating the meet of the current focus and the attribute concept of the new selection.

Definition 7 (see Section 2.1.3) shows how current selections can be considered as the query in the lattice and the extent of the focus concept then provides the result as described by Lindig [97]. However, since we are interested in a step-wise navigation algorithm and not simply the result of a single query we compute (and use) the focus on each navigation step. While the computation mechanism (see Theorem 8) is the same as the one used by Lindig [97] we have a different information need in that we do not only require the extent of a single concept (i.e., Lindig’s query result).

Figure 3.1 shows how the meet operation is used to calculate the new focus on the selection of two attributes in a small lattice example derived from a wine review dataset.

Items available for selection can be restricted to those that have a non-bottom meet with the focus, ensuring that a selection never returns an empty extent. The set of selections that do not cause the meet to go to bottom are called “significant keywords” by Lindig [97] and shown in Definition 9.

### 3.2.2 Refinement De-Selection

Intuitively the de-selection of the most recently selected item should return the focus concept to its previous position, undoing the selection. However, the lattice join does not undo a previous selection as computing the join of the focus with the attribute concept of the new de-selection will cause all previous selections to be removed, except the attribute we are de-selecting, which is counterintuitive. Therefore, in order to reverse a single selection operation we need to recalculate the focus as the meet in the lattice from all *still selected* items.

Our deselection performs essentially the same operation as illustrated by Lindig [98], although Lindig optimizes this operation by making use of the search path in order to calculate the new focus concept. Note that deselections do not always need to take place in the same order as the initial selections. The de-selection

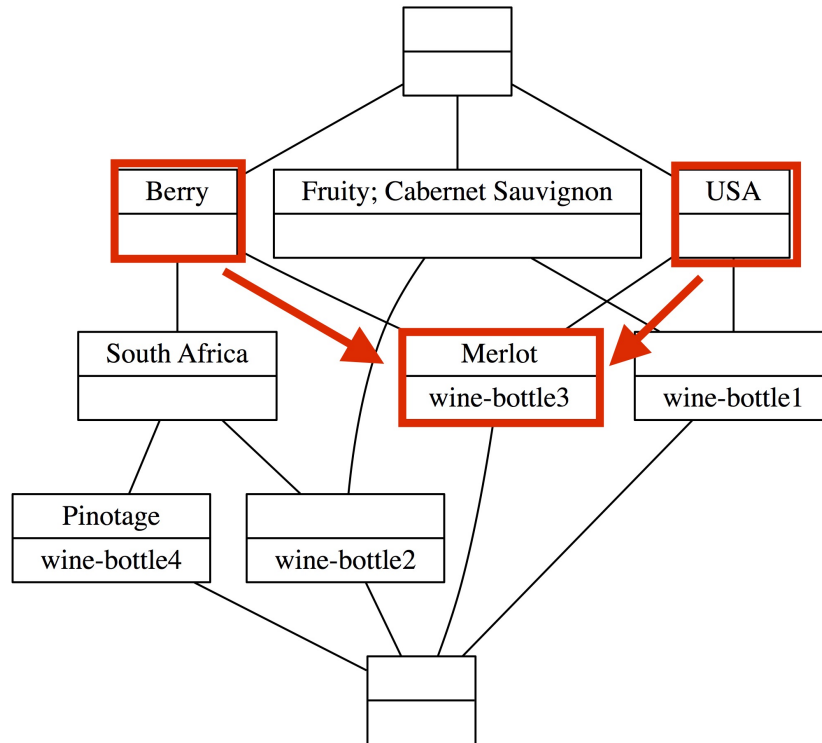


Figure 3.1: The meet of the concept introducing USA and the concept introducing Berry. The meet operation functions as conjunctive query and moves the focus concept down in the lattice.

operation can thus navigate to a focus which has not been visited during the previous navigation steps. De-selection is therefore not a strict undo operation. In this way our navigation operation indirectly supports pivot navigation by allowing the user to pivot from one tag selection to another by first selecting several tags and then removing one to pivot to the other tag selection. Figure 3.2 shows an example of an update to the focus concept on the de-selection of an attribute.

### 3.3 Broadening Navigation

There are several existing operations that extend the query result and can be considered as “broadening” navigation operations.

- We can de-select a previously selected term (as described in Section 3.2.2); under a purely conjunctive query interpretation the new focus is then computed as the meet of the introducing concepts of the remaining terms. Remember that the new focus has not necessarily been visited during the previous navigation steps, but it is a super-concept of the old focus, and conjunctive navigation with selection and de-selection is still functional and commutative.

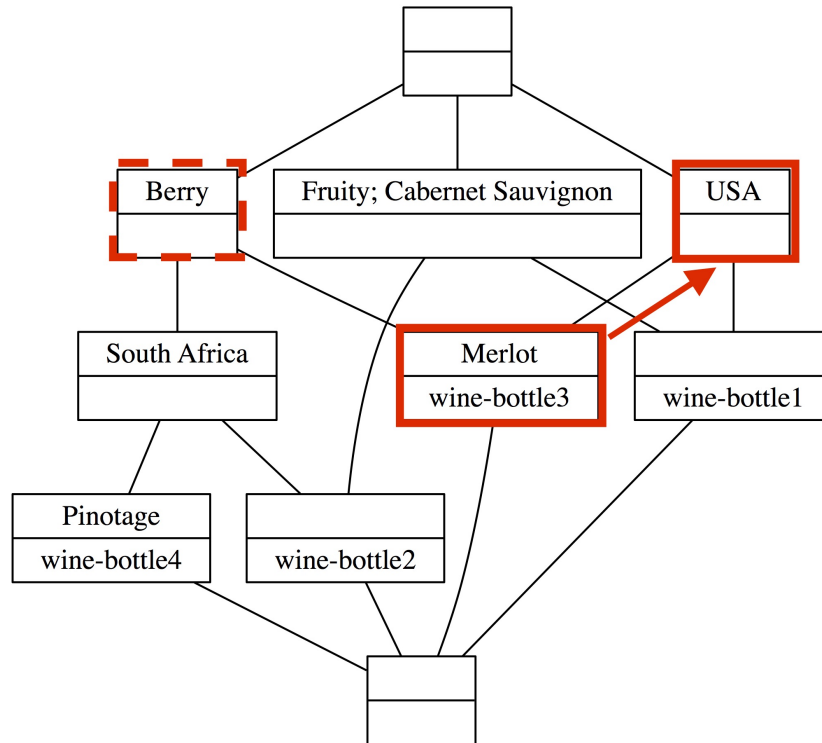


Figure 3.2: Update of the focus concept (the attribute concept of Merlot c.f. Figure 3.1) on the de-selection of the attribute Berry. The de-selection operation supports pivoting from the concept introducing Berry to the concept introducing USA, via the meet concept (Merlot selection).

- We can use a separate concept to represent each argument of an OR operator; the result of such a disjunctive query is then the union of all corresponding extents [120]. However, this disjunctive navigation gives up the single focus property and is no longer commutative, since the order of AND and OR operators matters.
- We can also retrieve or insert a *query concept* [47] into the lattice, where the query concept's intent contains the current search terms and consider the parents of the query concept (called the *query generator*) as a generalization [53]. Additionally, more children of the query generator can be included to broaden the results further. These are referred to as *cousin concepts* [53].
- We can use the lattice's join operation as a generalization operation; if the generalized concepts are determined by objects (rather than attributes) this is also known as *object-based navigation* [74]. This navigation has the single focus property by construction, and is still functional and commutative (when it is not mixed with refinement operations), but does not implement the Boolean



OR operation: due to the closure operations in the lattice construction, the extent of the new focus typically contains additional objects.

For contexts built from semi-structured data the attribute set is typically wide but there are few attributes for each individual object in the context, which leads to a sparse context. For example, a context constructed from the JUnit repository [17] has 2120 objects (i.e., revisions in the repository) with a total of 6465 attributes and an incidence relation of size 71688. Each object has an average of 33.8 attributes, equivalent to 0.5% of the total attribute set. On average a single attribute applies to 11 objects in this context, assuming equal distribution of attributes to objects which is unlikely in practice and so this is purely an approximation. The number of different combinations of possible attribute sets of size 33.8 (the average number of attributes for each object) from the total attribute set of 6465 is  $4.017 \times 10^{90}$ . However, in this context we can have a maximum of only 2120 different attribute set combinations as there is an upper bound imposed by the number of objects in the context (here 2120) since each object can have only one combination of attributes. In this context, where attributes apply to an average of 11 objects and each object has an average of 33.8 attributes an object shares a single attribute with an average of approximately 372 other objects. If we select two objects from our full object set at random then the probability that these two share an attribute is approximately 0.031 ( $\binom{372}{2} \div \binom{2120}{2}$ ) in this context. Therefore the probability of choosing two objects that share a selected attribute in this context is very low.

Note that this is still a relatively small context for our purposes. The concept lattice derived from this context has 133398 concepts, which is small compared to that of a repository with many more revisions (e.g., the Linux repository of over 500000 revisions [23]). For larger datasets with larger, sparser contexts, the probability of choosing two objects that share a selected attribute will only decrease.

Therefore, in these sparse contexts, the intersection of attribute sets from two concepts (used to compute the join) is narrow and often empty since there is no commonality between the attribute sets of the two concepts. Hence, the join operation often causes the focus to move back to top in the lattice. We refer to this as *overgeneralization*. The use of the join operation is particularly prone to overgeneralization in contexts where the attributes represent different categories or *facets* [118]. In particular, if we have functional facets (where each object can have only a single attribute for a given category, such as vintage of a wine), the join will effectively cancel the selected attributes from this category.

For example, in the concept lattice represented in Figure 3.1, if we calculate the join of the introducing concepts of two attributes from the functional facet country, South Africa and USA, then we see that the join of these two concepts would be top

since they have no other attributes in common. However, if we calculate the join of the introducing concepts from two attributes, Pinotage and Merlot, then the join of these two concepts is the introducing concept of Berry which they have as a common attribute. The join operation is thus unsuited as an intuitive generalization operation for the context tables generated from semi-structured data collections with functional facets. We therefore investigate an alternative generalization operation that makes use of only subsets of the extents of the attribute concepts of the selected items, in order to provide a more intuitive broadening navigation.

However, since not only one type of broadening navigation can be used to satisfy all use cases (i.e., Boolean disjunctive selections, generalizations from an attribute selection and generalizations from a selected object) we investigate three different approaches that each maintain the single focus property so that different navigation styles can be used interchangeably. We develop an approach for supporting disjunctive selections with a single focus concept (see Section 3.4). We also investigate generalization from attribute selections (see Section 3.5) and generalization from a single selected object (see Section 3.6).

### 3.4 Boolean Disjunctive Selection

The meet operation in the lattice provides us with a boolean AND operator. The meet of the attribute concepts of two items  $a$  and  $b$  (i.e.,  $\delta(a)$  and  $\delta(b)$ ), results in a concept whose intent contains both items  $a$  and  $b$ . However, Boolean OR navigation, where an attribute must only apply to *at least one* object is not supported by either the meet or join operations.

There has been previous work on supporting single disjunctive queries (but not step-wise navigation) in concept lattices. Priss [120] makes use of a Boolean disjunctive query operation which returns the union of the extents of the concepts that are retrieved for each of the items in the query when selected individually. However, this approach requires more than one concept to represent the query's result.

We instead interpret a Boolean disjunctive selection as a kind of conceptual scale [76] in the context table. Conceptual scales have been used for multi-valued attributes (e.g., review star ratings) where attributes are grouped in the context tables. For example, we might use an attribute **three stars or more** for all objects with three, four and five star ratings. This approach can also be applied to attributes such as the price of an object, where instead of using the actual object price as an attribute we create price groupings (e.g., **\$90-\$100**) so that objects of similar price groups will share a common attribute in the lattice even though their price might not be exactly the same. Traditionally, conceptual scales are pre-defined and the

scale is used to generate the context table. However, instead of using pre-defined scales in our approach, our scale is generated automatically when the user makes a Boolean OR selection of an item in the dataset. The scale is therefore created interactively and we update the context and thus the lattice on-the-fly. We follow this approach so as not to make our lattices much larger. Additionally, the presence of many disjunctive attributes would degrade the information in our lattices and add noise to the data.

In order to approximate a Boolean disjunctive selection with a single focus we therefore alter the underlying context table on-the-fly when a disjunctive selection is requested by the user. Our approach is illustrated in Figure 3.3, where the attribute Cabernet Sauvignon OR Merlot has been added to the context table and therefore the concept lattice. All objects with attribute Cabernet Sauvignon or attribute Merlot are given Cabernet Sauvignon OR Merlot as an additional attribute. The original attributes Merlot and Cabernet Sauvignon still remain in the context table and lattice.

In terms of the navigation steps, on Boolean OR selection of two attributes (e.g., a and b) we add the disjunctive attribute as an additional attribute to the context table and calculate the introducing concept of the newly created item ( $\delta(a \text{ OR } b)$ ) as the new focus concept in the lattice. Note that our approach returns the same query results as those that would be obtained in [120] for a *single* disjunctive query with no consequent navigation steps. However, it retains the single-focus property which has the advantage that further navigation steps can be applied following the boolean disjunctive selection.

For example, if we select the attribute Merlot and then add Cabernet Sauvignon, our new focus becomes the attribute concept of Cabernet Sauvignon OR Merlot in the lattice presented in Figure 3.3. If we then select the attribute South Africa then we navigate to the object concept of wine-bottle2 which now has as attributes Fruity, Cabernet Sauvignon, Berry, South Africa and Cabernet Sauvignon OR Merlot. We see here that the attribute Cabernet Sauvignon OR Merlot should no longer be in the intent because according to the absorptive laws of propositional logic (i.e.,  $S \cup (S \cap T) = S$ ) the combination of the attributes Cabernet Sauvignon and Cabernet Sauvignon OR Merlot should result in only the attribute Cabernet Sauvignon.

Therefore, in lattices in which we have introduced disjunctive attributes the intents of the subconcepts of the attribute concepts of a disjunctive attribute need to be calculated by applying the absorptive laws to remove redundant disjunctive attributes.

If the original lattice and the lattice after the disjunctive attribute has been inserted are isomorphic then we see that the insertion of the disjunctive attribute has not resulted in an additional concept in the lattice but only in the addition of

the disjunctive attribute to a concept already contained in the lattice. In this case we know that there is another attribute in the context which is equivalent to the newly inserted disjunctive attribute. In order to test whether an additional concept will be introduced into the lattice when we insert the disjunctive attribute into the context table we can check whether  $\delta(a) \vee \delta(b)$  is equivalent to  $\delta(a \text{ OR } b)$ . If the join of the attribute concepts of both  $a$  and  $b$  (i.e.,  $\delta(a) \vee \delta(b)$ ) has an extent equivalent to  $\pi_O(a) \cup \pi_O(b)$  then we see that  $\pi_A(\delta(a) \vee \delta(b))$  is equivalent to  $a \text{ OR } b$ . Therefore, our approach also allows users to discover additional information about the dataset.

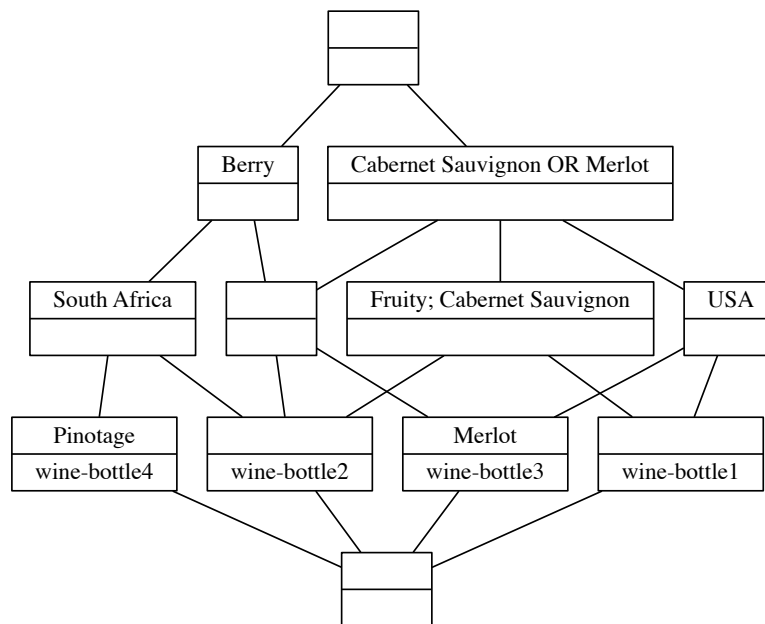


Figure 3.3: Full concept lattice generated from the wine data context in Figure 2.1; where attributes *Cabernet Sauvignon* and *Merlot* have been selected for Boolean OR navigation. A new concept with attribute *Cabernet Sauvignon OR Merlot* has been inserted into the lattice. The concept has been inserted at the atom level because there are no objects which have the combination of attributes that are used in the query.

Codocedo et al. [53] also use an approach where concepts are inserted into the lattice on-the-fly. In their approach the *query concept* is inserted into (or identified in) the concept lattice with a placeholder object (i.e., the query object) and all the attributes that form a part of the current query [47]. Note that if the query concept cannot be identified in the lattice and needs to be inserted, the query concept (with extent formed of the placeholder query object) will always appear at the atom level

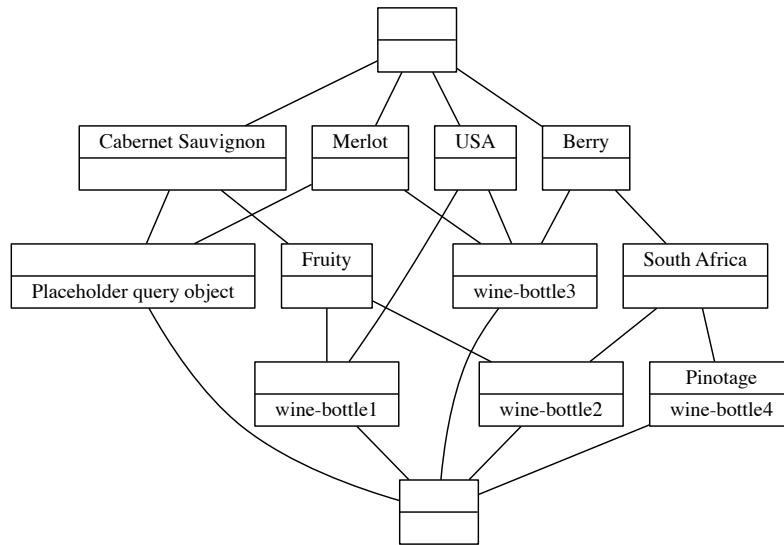


Figure 3.4: Query concept with intent Merlot and Cabernet Sauvignon inserted into the concept lattice. The extent is a placeholder object used to represent the query.

of the lattice as there does not exist another object that possesses all the attributes forming the current query (cf., Figure 3.4).

The superconcepts of the query concept are then referred to as the *query generator*. The *cousin concepts* of the query concept refer to the other immediate subconcepts of the *query generators*. The cousin concepts and the query generators are used to implement a broadening approach in the concept lattice [53]. The query's result is then returned as the union of the cousin concepts' extents. Using this approach the results of a navigation step are also returned from more than one concept in the lattice causing the navigation algorithm to have more than one focus.

This approach (illustrated in Figure 3.4) does not implement a purely disjunctive query operation as the query generators (i.e., the superconcepts of the query concept) are not necessarily the attribute concepts of the items selected for a disjunctive query. Therefore, the insertion of the query concept does not satisfy Boolean disjunctive navigation in the same way as the insertion of an additional concept with a disjunctive attribute does in our approach.

### 3.5 Attribute-Oriented Generalization

The Boolean OR based navigation results in an additional attribute being added to the context table and changes the concept lattice. The join operation supports

broadening navigation, but, if the extents of both concepts are large then the join is likely to overgeneralize and can result in the top concept ( $\top$ ), thereby losing all previous navigation steps and resulting in a low precision for the constructed query.

To facilitate a less strict generalization which results in the focus being below the Boolean OR selection we compute the join from only a *subset* of the objects in the full extents of the two concepts selected for broadening navigation. For example, if we want to determine what wine bottles from two different countries (USA and South Africa) have in common we might find that there are no attributes common to *all* of the wines produced in both countries. However, if there is a property that is common to *some* of the wines from from each of the individual countries, then we are interested in navigating to the concept expressing this information instead of navigating to the top of the lattice. While the insertion of the additional disjunctive attribute South Africa OR USA would allow us to navigate to a concept which has as extent all wines from these two countries, there may be no other attribute which is common to all of these wines and therefore the Boolean OR navigation (discussed in Section 3.4) does not provide a mechanism for finding out what attributes *most* of the wines from these two countries have in common. We therefore use an approximation to facilitate an attribute-oriented generalization which is less strict and will allow us to discover if there is an attribute which is common to *some* of the wines from each of the two selected attributes for broadening.

### 3.5.1 Generating Candidate Focus Concepts

Our broadening approach results in an updated focus that shows attributes which are common to *some* of the objects in the current focus and *some* of the objects in the attribute concept of the new item ( $\delta(\mathbf{b})$ ) selected for broadening. Since we need to retain the single-focus property on each navigation step to ensure that our different navigation styles can be used interchangeably, the main challenge of this approach is to determine a suitable heuristic that controls which objects are present in the new focus concept.

In order to calculate the new focus after a generalization selection we traverse the lattice with a depth-first approach, using the focus as starting point. For each not yet visited concept in this traversal we then also perform a depth-first traversal starting at  $\delta(\mathbf{b})$ . We compute the join of every concept derived from these iterations as candidate focus concepts for the next navigation step.

This process is equivalent to computing the pairwise joins of all concepts in the ideal of the initial focus with the ideal of the attribute concept of the new selection.

The set of candidate focus concepts is computed as

$$\{X \vee Y \mid X \leq Focus, Y \leq \delta(b)\}$$

. Each of the candidate focus concepts contains properties that are common to some of the objects in the initial focus and some of the objects in the concept introducing the new selection ( $\delta(b)$ ). However, there could be many resulting candidate focus concepts and we therefore need to select a new focus from this pool in order to maintain only a single focus.

### 3.5.2 Selecting a new Focus Concept

We can use heuristics to calculate which concept from the pool of candidate focus concepts to return as the new focus concept in our single-focus navigation.

We define a heuristic that tries to return the concept with the largest extent such that at least one object from the extent of the initial focus is present and at least one object from  $\delta(b)$  is contained in the extent. Note that using these criteria, if the join which is not the top concept was part of the pool of candidate focus concepts then the join would satisfy these criteria and would be returned as the new focus concept. Therefore, if the join is not the top concept we do not perform the computation at all, we simply return the join concept.

In our approach we return the concept  $c$  ( $c = (X, Y)$ ) from the pool of candidate focus concepts which results in the highest score where the score is calculated as

$$score = |X| - \left| |\pi_O(Focus)| - |\pi_O(\delta(b))| \right|, \text{ where } |\pi_O(Focus)| > 0 \text{ and } |\pi_O(\delta(b))| > 0$$

. The score is a heuristic that is calculated so as to favor concepts with larger extents that have a similar number of objects from both the attribute concept of the new selection and the previous focus. If there are multiple concepts that have the same score then we can return from these the concept with the largest extent.

Using this heuristic our broadening operation therefore generalizes as much as possible without losing all previous selections and navigation steps (navigating to the top element).

Another heuristic we can use to select the new focus concept from the pool of candidate concepts is the  $F_2$  score from Information Retrieval [128]. The  $F_2$  measure is a combined measure of both precision and recall where recall is more highly weighted. The score is given as  $F_2 = 5 \times ((precision \times recall) \div ((4 \times precision) + recall))$ .

### 3.5.3 Worked Example

We present a worked example that demonstrates how the new focus concept can be selected from the pool of candidate focus concepts using both our heuristic (described above) and the standard  $F_2$  measure from Information Retrieval.

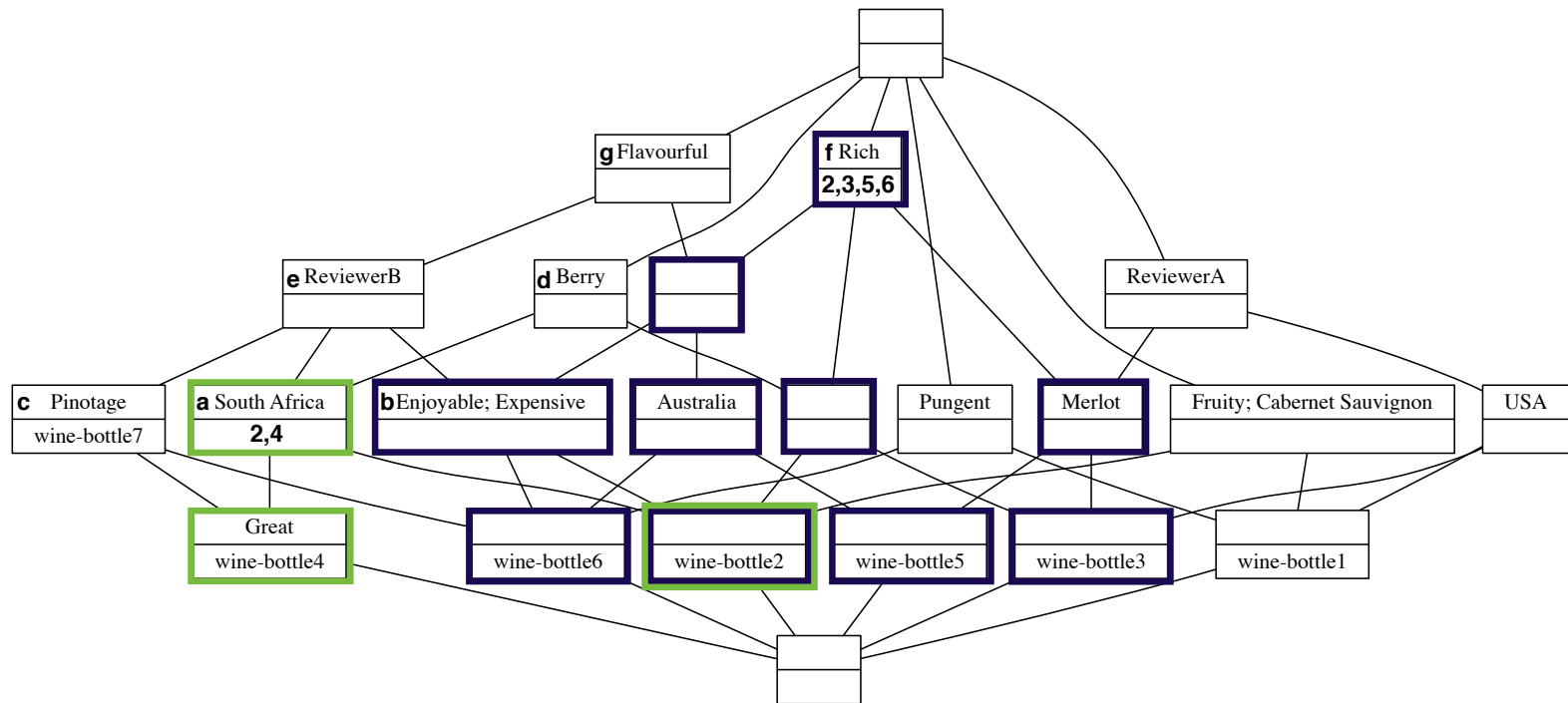


Figure 3.5: Extended wine context which depicts the sub-concepts of the current focus (South Africa) in green and the sub-concepts of the new selection for generalization (Rich) in black. We compute the pairwise join of all sub-concepts of the attribute concepts from both Rich and South Africa to generate candidate focus concepts on an attribute-oriented generalization selection. We then select one of these concepts as the new focus based on a heuristic.



We have added additional attributes and objects to our original wine context to produce a larger lattice (see Figure 3.5) which better illustrates our approach. Our initial focus is the attribute concept of **South Africa** (i.e.,  $\delta(\text{SouthAfrica})$ ) from which we want to generalize with the attribute concept of **Rich** (i.e.,  $\delta(\text{Rich})$ ). However, when we compute the join  $\delta(\text{SouthAfrica}) \vee \delta(\text{Rich})$  the result is top because there are no attributes in this context which are common to wine-bottle2, wine-bottle3, wine-bottle4, wine-bottle5 and wine-bottle6. Note that if we were to insert the Boolean OR attribute for **South Africa OR Rich** then this concept would have as extent wine-bottles 2-6 and the join would no longer go to top. However, using this approach we are not guaranteed to discover if there are any attributes common to *most* of the wine bottles 2-6 as is the goal of this particular generalization. While the wines 2-6 would have as attribute **South Africa OR Rich** this would not give us any additional information about the wines. Therefore we compute all the pairwise joins of the subconcepts of  $\delta(\text{SouthAfrica})$  with the subconcepts of  $\delta(\text{Rich})$  in order to generate a set of candidate focus concepts. Table 3.1 shows all the extents of concepts in our pool of candidate focus concepts and for each concept the size of the extents of the candidate focus concept and the two concepts used to compute it are given as well. We see that the attribute concepts of **Flavorful** and **ReviewerB** both have a score of four, however, the attribute concept of **Flavorful** will be the new focus as this has the largest extent. The attribute concept of **ReviewerB** has obtained a high score because even though it has an extent of only four it contains an equal number of objects from the initial focus ( $\delta(\text{SouthAfrica})$ ) and the newly selected concept ( $\delta(\text{Rich})$ ).

We see from this example that while there were no concepts (other than top) that had as extent wine-bottles 2-6, (which the disjunctive selection discussed in Section 3.4 would have returned) there is a concept which has as extent wine-bottles 2,4,5,6 and 7 (80% of the bottles we are looking at) and all these bottles share the attribute **Flavorful**. Therefore, our approach has allowed us to generalize to the attribute concept of **Flavorful** instead of top. The attribute concept of **Flavourful** provides us with an 80% precision and an 80% recall. This approach uses existing attributes and is thus better for understanding the dataset than the disjunctive navigation discussed in Section 3.4.

### **$F_2$ -Measure**

We can also use the F-measure [143] as the heuristic for selecting the focus concept from the pool of candidate focus concepts. The F-measure uses a weighted mixture of the precision and recall values to calculate a score for evaluation in information retrieval. The  $F_2$ -measure is a version of the  $F_\beta$  measure which places more emphasis on recall than precision. Using the  $F_2$ -measure we see that the attribute concept of

Table 3.1: Calculated scores for candidate focus concepts generated when generalizing the current focus concept ( $\delta(\text{SouthAfrica})$ ) with the new selection of attribute Rich using the heuristic introduced in Section 3.5.2.

Objects (wines)	Attributes Introduced at Concept	Size of Concept Extent ( $ C $ )	$ A $	$ B $	Score for Concept
a) {2,4}	{South Africa}	2	2	1	1
b) {2, 6}	{Enjoyable, Expensive}	2	1	2	1
c) {4, 6, 7}	{Pinotage}	3	1	1	3
d) {2, 3, 4,}	{Berry}	3	2	2	3
e) {2, 4, 6, 7}	{ReviewerB}	4	2	2	4
f) {2, 3, 5, 6}	{Rich}	4	1	4	1
g) {2, 4, 5, 6, 7}	{Flavorful}	5	2	3	4

Rich would be selected as the new focus concept. Our  $F_2$  scores are calculated as in Table 3.2. While the attribute concept of Rich might have a high precision and recall this concept was not scored favorably using our heuristic (see Table 3.1) because it contains an unequal balance of objects from the initial focus (attribute concept of South Africa) and the new selection (Rich).

Table 3.2: Calculated scores for candidate focus concepts using the  $F_2$ -measure when generalizing the current focus concept ( $\delta(\text{SouthAfrica})$ ) with the new selection of attribute Rich.

Objects (wines)	Attributes Introduced at Concept	Precision	Recall	$F_2$ -Measure
a) {2,4}	{South Africa}	1	0.4	0.454
b) {2, 6}	{Enjoyable, Expensive}	1	0.4	0.454
c) {4, 6, 7}	{Pinotage}	0.67	0.4	0.435
d) {2, 3, 4}	{Berry}	1	0.6	0.652
e) {2, 4, 6, 7}	{ReviewerB}	0.75	0.6	0.625
f) {2, 3, 5, 6}	{Rich}	1	0.8	0.833
g) {2, 4, 5, 6, 7}	{Flavorful}	0.8	0.8	0.800

### 3.6 Object-Oriented Generalization: Finding Similar Objects

An operation to find similar objects allows generalization from a single object, to a concept containing more objects (i.e., with a larger extent) where the objects share

attributes and could be considered similar. While Section 3.5 has discussed an approach to generalize by selection of an additional attribute, this section discusses an approach to generalize from a single selected object. If the user has navigated to a concept with only one object in the lattice (e.g., a specific wine) then a generalization operation can be applied to move the focus higher in the lattice to a concept containing similar objects without the need for the user to try to select all combinations of attributes for the specific object until one shifts the focus higher in the lattice. All concepts in the lattice in which the object of interest appears in the extent (i.e., the superconcepts of the concept of interest) which do not have an empty intent can be considered to present similar objects since the objects share at least one common attribute. However, since we need to maintain the single-focus property so that different navigation styles can be applied interchangeably we need a method to select one of these concepts as the new focus concept.

To find objects that are similar to a selected object in the dataset we introduce a *more-like-this* operation. For example, if we want to find bottles of wine that are similar to a bottle that we have previously tried, we can apply the *more-like-this* operation to shift our focus to a concept that contains similar bottles, without the user needing to be aware of any of the attributes of the wine.

In multi-faceted data, we may be interested only in objects that share attributes of a particular facet. For example, in the wine data we might have as an attribute the name of the reviewer for each wine bottle. However, when considering similar wine bottles we are more likely to be interested in the review text and varietals than that the wine has been reviewed by a specific reviewer. We therefore look at attributes of the object of interest only from a subset of the full set of facets in cases where not all facets of attributes are useful for comparing objects. The facets that should be excluded from the *more-like-this* operation need to be manually specified by the user. Note that this is similar to the standard feature selection approach that is used when performing a nearest neighbor calculation [42] where not all features are weighted evenly and some features are excluded (i.e., given a zero weighting) [86]. While some attributes of our data might be included in the context table to provide additional paths along which to navigate, these attributes might need to be excluded from a calculation of similar objects in the context.

Calculating the size of the extent of the attribute concept can provide an Inverse Document Frequency [139] measure for the attribute. If the extent of an attribute concept is large, then the objects in the extent are unlikely to be very similar, since the attribute is common to a large percentage of the objects in the full corpus. For example, in the wine data we might have attributes **red** and **white** which each apply to half of the objects in the context. We might also have a flavor description such

as *berry* which only applies to five percent of objects in the context. Therefore, the attribute *berry* can better allow us to find similar objects than the attribute *red* because it is more specific (characterizes a smaller percentage of the objects in the context).

In our approach we consider the full attribute set for the object of interest. We then remove all attributes that are of undesirable facets (such as wine reviewer in the context of wine data). For each attribute we calculate the size of the extent of the attribute's introducing concept, which is used as a measure for how well the attribute characterizes the object of interest. We then remove the attribute whose introducing concept has the smallest extent and calculate the meet of the attribute set. Since attributes whose attribute concept has a small extent are most descriptive we want to use these attributes to find similar objects. However, these attributes might also uniquely describe the object of interest (i.e., the size extent of the attribute concept is one) in which case these attributes need to be excluded from the full attribute set used to find similar objects since there are no other objects with the specific attribute. We therefore remove from the attribute set the most specific attributes for the object of interest so that the attribute set is not necessarily unique to the object of interest anymore. By removing the most specific attribute from the attribute set, the meet of attributes is able to move up in the lattice since the attribute set now consists of more general attributes that are introduced higher up in the lattice. If the meet of the attribute set is not the object concept of the object of interest (if the meet concept is now higher up in the lattice), then the extent of this concept will contain objects which are similar to the object of interest. However, if the extent of the meet of the attribute set only contains the object of interest then we again remove the attribute whose introducing concept has the smallest extent (the most specific attribute). We apply this process iteratively until the meet of the attribute set is no longer the object concept of the object of interest. The resulting concept contains other objects which are similar to the object of interest because they share *some* of the properties (of chosen relevant facets) of the object of interest.

### Worked Example

We can use the same lattice as presented in Figure 3.5 to illustrate a generalization from a particular object. If we generalize from *wine-bottle4*, we have as attributes *Great*, *South Africa*, *Berry*, *Pinotage*, *ReviewerB* and *Flavorful*. However, if we exclude the wine reviewer facet from the similar objects calculation then we do not consider the attribute *ReviewerB*. For each attribute we first calculate the size of the extent of its introducing concept in order to get an indication as to how specific the attribute is. Table 3.3 presents the sizes of the extents for each attribute.

Table 3.3: The attributes for wine-bottle4 and the corresponding sizes of the extent of their attribute concepts. Note that ReviewerB is not shown in this table as the facet wine reviewer has been excluded from the calculation of similar objects.

Attribute	Size of extent of attribute concept
Great	1
Pinotage	3
South Africa	2
Berry	3
Flavorful	5

Since the attribute **Great** applies only to wine-bottle4 we first remove this attribute from the meetset in order to generalize. However, after removing **Great** the meet of all the attributes in the set is still only wine-bottle4. Since the attribute concepts of **Pinotage** and **South Africa** both have an extent size of two, we need to recalculate the meetset by excluding each of these attributes. The meet of the attribute concepts of **South Africa**, **Berry** and **Flavourful** is the attribute concept of **South Africa** which has wine-bottle2 and 4 in its extent. The meet of **Berry**, **Flavourful** and **Pinotage** is still only wine-bottle4. We therefore navigate to the attribute concept of **South Africa** which indicates that wine-bottle4 can be considered the most similar to wine-bottle2 according to our parameters. The new focus concept is therefore  $\delta(\text{SouthAfrica})$ .

### 3.7 Conclusion

In this chapter we have described our single-focus refinement and broadening operations in concept lattices. Our navigation uses a single focus concept on all navigation steps so that refinement and broadening operations can be used interchangeably. Our refinement navigation is achieved using the standard meet operation and our deselection operation calculates the meet of the attribute concepts of all still-selected attributes. Therefore, the deselection operation does not only function as an undo operation. We support boolean disjunctive navigation by adding new disjunctive attributes into the context table and therefore concept lattice, in order to allow disjunctive navigation using only a single focus. Our attribute-oriented generalization results in a new focus which contains some of the objects from the initial focus and some of the objects from the newly selected concept so as not to navigate to top when there is no attribute that applies to all of the objects in the extent of the initial focus and the newly selected concept. We also introduce a method of finding similar objects from one selected object by considering only attributes of relevant facets.

## Chapter 4

# ConceptCloud Tool

In this chapter we describe the ConceptCloud tool in which we have implemented the approach described in the previous chapters. The tool supports the construction of concept lattices and tag clouds from XML and JSON data that is provided as input. ConceptCloud provides various interface customizations, such as allowing multiple interlinked tag cloud views to be generated. Each tag cloud is individually customizable, and allows users to perform operations such as filtering the tag cloud view, limiting the number of tags that are displayed and sorting the tags in the cloud. We have also developed a scripting language for ConceptCloud called ConSL which allows views to be scripted so that they can be saved and reloaded again at a later stage.

ConceptCloud comprises three main components that extract meta-data from an archive, construct a context table in the desired format, and display the tag cloud of the resulting lattice which supports navigation as described in Chapter 3. We describe our approach for constructing a tag cloud directly from a concept in the lattice in Section 4.1. Once a tag cloud has been constructed navigation in the tag cloud is driven by selections and de-selections of tags, which causes a recalculation of the focus concept in the concept lattice. All front-end filtering operations are dynamic and so users can quickly alter the amount of information that is presented in the tag cloud views. The browser is generic and can show tag clouds of different context types. It is also completely automatic: there are no manual pre-processing steps, and the user only needs to point ConceptCloud at the data source. The generated context tables can be saved in XML format so that they can be loaded again without extraction.

While this section discusses the implementation details of the ConceptCloud browser, an application of ConceptCloud to software version control repository data is shown in Section 5.5 and an application example to academic publication data is shown in Section 7.5. Performance metrics of the browser when used to analyze

different sized software repositories is presented in Section 5.6.

## 4.1 Tag Cloud Visualization

### 4.1.1 Tag Cloud from a Concept

We visualize semi-structured data with a tag cloud (see Section 2.3 for more discussion on tag clouds) that we construct from the focus concept in the lattice. Since a concept comprises a set of objects and a set of attributes, it is tempting to use the attributes (i.e., the intent) as the tag cloud. However, this produces degraded clouds because (i) the intent only contains the attributes common to all objects, and (ii) each attribute only occurs once so that all tags would have the same size. Instead, we use the intents of the extents; more precisely, we collect all attributes of the defining concept of each object in the extent of the focus concept; we also add the objects themselves, to allow their direct selection via the tag cloud. Note that tags derived from the intent of the focus concept will be sized the largest in the tag cloud because they appear in every concept (i.e., all object concepts from the objects in the extent of the current focus) that is used to derive the tag cloud. Other tags that are not part of the focus's intent are sized according to the proportion of objects in the extent to which they apply.

We say that a tag in the cloud is *implied* if it has not been selected explicitly, but corresponds to an attribute in the focus's intent. Implied tags reveal the data's internal structure, similar to the way association rules reveal the implicit structure of shopping baskets [156] but without any additional cost.

**Definition 10** *The tag cloud from a concept  $c = (O, A) \in B(\mathcal{C})$  is defined as  $\tau(c) = O \uplus \biguplus_{o \in O} \pi_A \sigma(o)$ .*

Here  $\uplus$  denotes multiset union. By construction, the objects in the tag cloud induce subconcepts of the concept from which the tag cloud was derived; moreover, all tags have a non-bottom meet with that concept.

For example Figure 4.1 shows which concepts would be used to construct a tag cloud when the attribute Alice is selected. We use the object concepts of the objects in the extent of the current focus concept (here revision1, revision2 and revision3) and the multiset union of the intents of all these concepts to construct the tag cloud.

**Proposition 11** *Let  $c \in \mathcal{B}((\mathcal{O}, \mathcal{A}, \mathcal{I}))$  be a concept,  $o \in \mathcal{O}$ , and  $t \in \mathcal{O} \cup \mathcal{A}$ . Then (i)  $o \in \tau(c) \Rightarrow \sigma(o) \leq c$ , and (ii)  $t \in \tau(c) \Rightarrow \delta(t) \wedge c \neq \perp$ .*

We make use of an alphabetically-sorted tag cloud layout because it simplifies textual search within the tag cloud. We scale each tag  $i$  between the given minimum

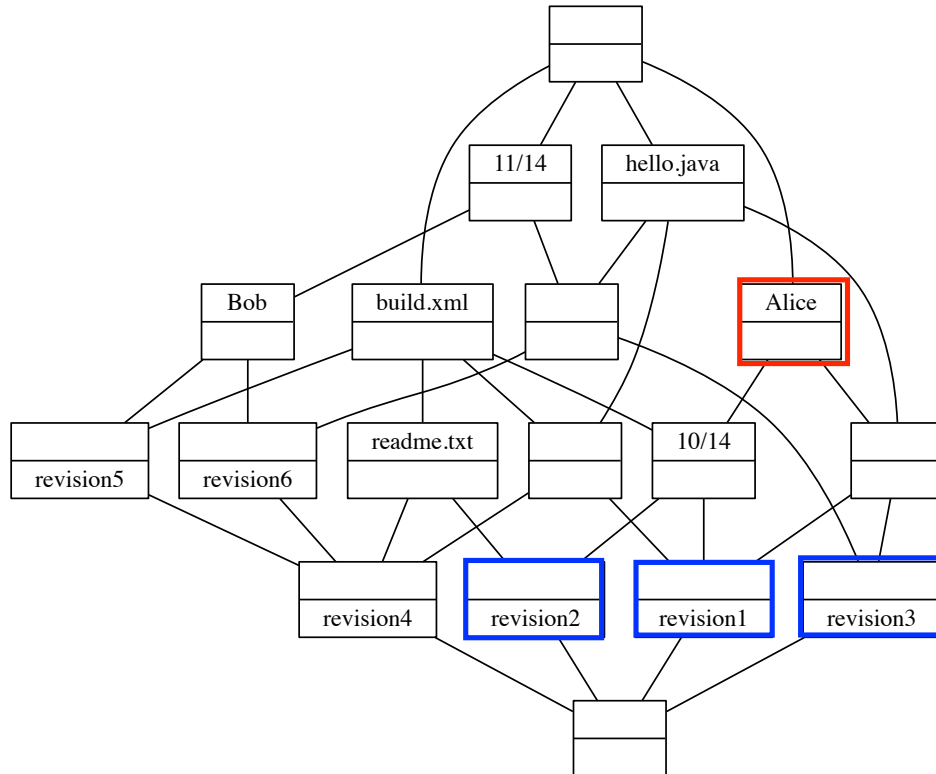


Figure 4.1: The concepts used to construct a tag cloud from a particular focus concept in the lattice. We use the multiset union of all attributes derived from the introducing concepts of the objects in the extent of the current focus. The current focus is indicated in red (Alice) and the concepts used to construct the tag cloud are indicated in blue (revision1, revision2 and revision3)

and maximum font sizes  $f_{min}$  and  $f_{max}$ , according to its weight  $t_i$  in relation to the minimum and maximum weights in the context table,  $t_{min}$  and  $t_{max}$ ; hence,

$$\text{size}(i) = \left\lceil \frac{(f_{max} - f_{min}) \cdot (t_i - t_{min})}{t_{max} - t_{min}} \right\rceil + f_{min} - 1$$

for  $t_i > t_{min}$  and  $\text{size}(i) = f_{min}$  otherwise.

Our tag clouds provide an intuitive interface to the underlying data contained in concept lattices. The concept lattices are generated directly from formal contexts which can be constructed from different types of data archives. Figure 4.2 shows an overview of our approach on data from a software version control repository. We see that the initial focus is the top concept in the lattice, which generates a tag cloud that contains all of the data in the underlying dataset as the top concept in the lattice includes all the objects in the dataset. Therefore, from the initial tag cloud all tags are available for selection. When tag Alice is selected then the focus is recomputed as the meet of the top concept in the lattice and the concept introducing Alice. However since the concept introducing Alice is directly underneath the top



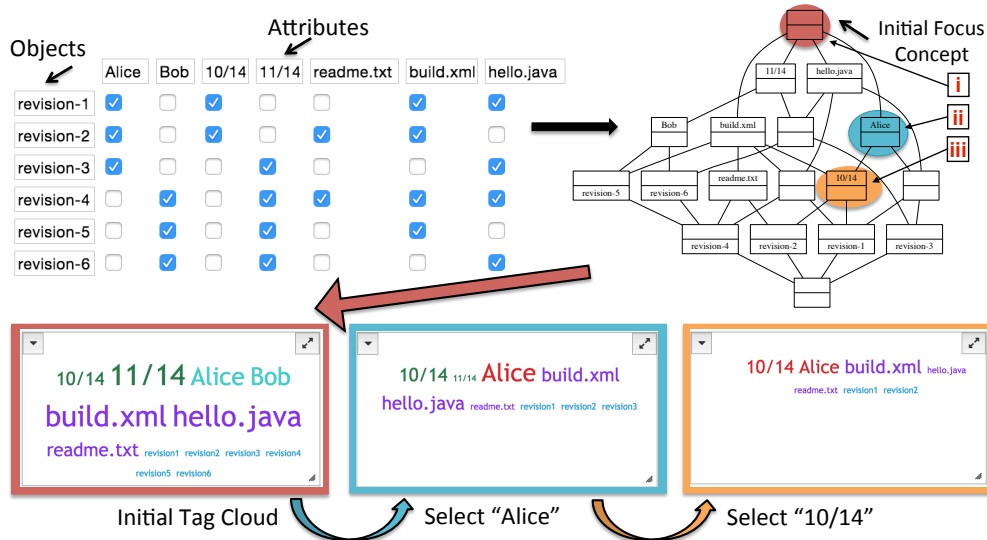


Figure 4.2: Navigating Concept Lattices with Tag Clouds: tag clouds correspond to the matching colored concepts in the lattice. Context table (top left) used to generate concept lattice (top right). Tag clouds are refined on each tag selection (selected tags shown in red).

concept in the lattice, the new focus is simply updated to the concept introducing the attribute Alice. The tag cloud is then updated to show only information pertaining to the selection of Alice. For this example, where the objects in the context table are revisions in a software version control archive, the tag cloud shows only information for all revisions that have been checked-in by Alice. The tag cloud is then further refined by selecting the date tag 10/14 which moves the focus even further down in the lattice and further restricts that tag cloud to showing only information from revisions which were checked-in by Alice on 10/14. The tags in the tag cloud can be de-selected in any order (meaning that is it possible to deselect Alice before 10/14) and so the de-select does not perform an “undo” operation (see Section 3.2.2 for further details).

#### 4.1.2 Relation to Information Retrieval

Our lattice-based browsing approach is related to classical information retrieval (IR) [102,143]. The context table itself can be seen as a Boolean version of the document-term matrix. Document-term matrices are used to represent how many times words appear in a document corpus. The document-term matrix can therefore be seen as a specific instance of a context table in which the objects are documents in a corpus and the attributes are the words contained in the document set. However, the context table will only indicate whether or not a word is present in a document

and would not indicate how many times the word appears in each document.

The concept lattice can be seen as a representation of the regular and inverted index in information retrieval, which map which words are present in a document and which documents make use of a specific word respectively. In the concept lattice the object concept for a specific document will have as intent all words which are present in the document. Alternatively, the attribute concept for a specific word will have as extent all documents in the full corpus in which the word is present.

In our approach, tags selected in the tag cloud can be seen as a conjunctive query to the full document set. The focus concept will then contain all words forming the query in its intent and the extent of the focus provides the result of the query as all documents in which the queried terms appear.

Our tag clouds can also be seen as presenting an aggregation of the Boolean term frequencies for each document in the query's result (i.e., extent of the current focus). By construction the tags in the tag cloud (which represent words in the documents) are scaled according to how many documents in the query's result the words appear in. For example, words that appear in every document in the query's result will be the largest in the tag cloud. Our tags are also scaled according to the size of the entire document corpus, so that tags on subsequent selections (after forming a query) can never be larger than the overview of the entire document corpus, presented in the initial tag cloud.

The concept lattice itself provides us with an efficient way to compute this tag cloud. If we were to store and make use of only the inverted index, we would first need to retrieve all documents indexed by the selected tags (which forms the extent of the current focus in our approach). We would then need to iterate over the entire vocabulary (the full attribute set in the context table) to compute the size of the intersection of each term's inverted index with the query's result, in order to identify which words appear in the document, forming the query's result.

If we were to optimize the IR-based implementation this would mean storing and using the information in essentially the same way as the lattice-based implementation. By using the concept lattice we have the advantage that we can exploit the lattice structure which allows us to update the focus incrementally when new selections are made and to show which other tags are implied by (i.e., always occur along with) the current selection set.

## 4.2 Tool Architecture

ConceptCloud has been developed using the Play Framework [26] which provides a Java web server. The back-end of ConceptCloud is written in Java, and is responsible

for creating the concept lattice and serving the tag cloud data.

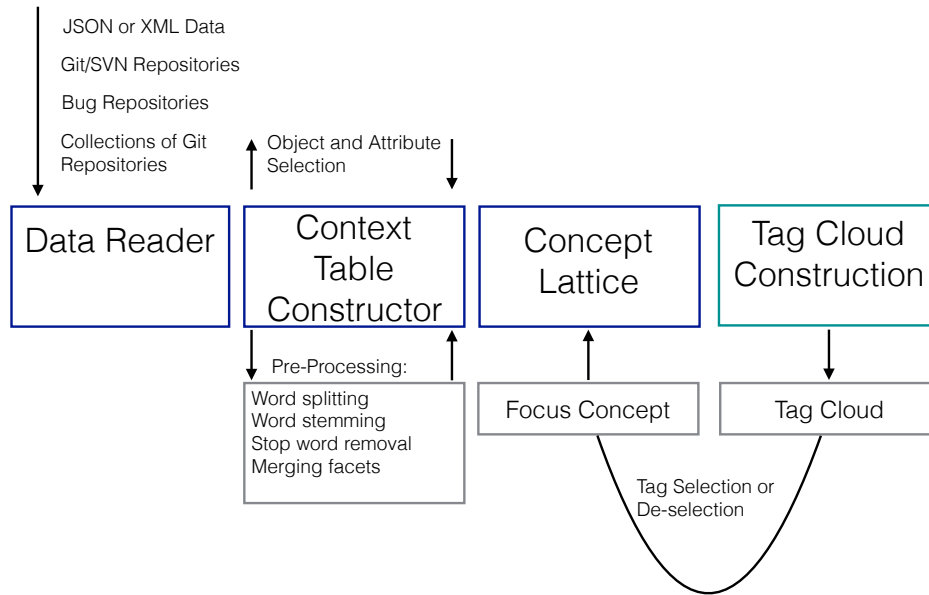


Figure 4.3: An overview of the main components of the ConceptCloud browser. Each component performs a specific function. ConceptCloud can read data in various formats and apply different pre-processing steps in order to construct context tables and therefore concept lattices. Tag clouds are constructed by a component directly from the current focus in the lattice and the focus is updated when selections are made in the tag cloud.

All filtering and viewer customizations are handled dynamically using Javascript, so that filtering operations can be performed on cached data without requiring further communication with the server. The core of the ConceptCloud framework comprises of the context table and concept lattice construction components as well as the support for navigation within the concept lattice (see Figure 4.3 for an overview). All visualization operations in the tool are generic and are applicable for any data source that has been loaded. For each new dataset the data can either be loaded in XML or JSON format, or a new data reader component can be constructed to allow a context table to be constructed directly from the new dataset.

We have written data reader components that can read data directly from Git and SVN repositories as well as bug tracking systems. Therefore, Git and SVN repository data as well as GitHub issue tracking data can be loaded directly into ConceptCloud for visualization. We have also written a data reader component that allows us to read data from a large collection of repositories hosted on GitHub. All other data sources are loaded through JSON or XML files.

### 4.2.1 Pre-processing Steps

The ConceptConstructor component of ConceptCloud allows users to load an XML or JSON file along with a Document Type Definition (DTD) describing the structure of the data file, and choose various pre-processing steps to apply to the dataset. Users are also able to choose how to build their context table, by choosing which field will be used as the object and which fields will be used as the attributes. Therefore, users are also able to construct more than one concept lattice from the same data source in different ways by using different object types.

ConceptConstructor internally represents the dataset as a connected graph structure which is then manipulated through the various pre-processing steps. The graph only stores the fields that will be used in order to construct the context table. The graph also stores the relationships between an object and its attributes, so even an attribute that applies to multiple objects only needs to be pre-processed once. When the user is satisfied with the combination of pre-processing steps, ConceptConstructor can generate an XML file of the context table which can then be loaded directly into ConceptCloud to be visualized.

Various pre-processing steps are provided as part of the ConceptCloud framework in order to generate tag clouds in which the information is visualized intuitively and where there is minimal repetition of tags in the cloud.

The word splitter splits free-text fields according to whitespace in the text so that the tags are formed from individual words rather than text paragraphs. Trailing punctuation is also removed from all words. The stemmer stems all words of a selected facet to reduce duplication of tags which have the same stem. We make use of the Porter Stemmer [116] in order to generate the word stems. However, since word stems are not necessarily proper words, we represent all words evaluating to the same stem by the most common word that evaluates to that stem. We also include a stop word removal component which can remove common stop words from selected facets. Users can also load their own stop word list in order to remove only a custom set of words. ConceptCloud also provides support for merging different categories (or facets) so that these will be presented together in the tag cloud. Each category of information is treated separately in the concept lattice and the tag cloud, so if two categories in the underlying dataset represent the same information then these can be merged in a pre-processing step.

### 4.2.2 Concept Lattice Construction

We use an incremental lattice construction approach which builds on the Colibri-Java implementation [85]. This approach provides an indexing structure for the data which records which objects apply to each attribute and which attributes apply

to each object. When concepts need to be constructed on the fly, the approach calculates for each set which attributes are common to all objects in a particular object set or which objects are common to a group of attributes in a particular attribute set. Therefore, this approach allows us to mitigate some of the large indexing times that are normally associated with formal concept analysis because we never need to generate the full lattice. Since we do not compute the full lattice we can also support on-the-fly scaling and disjunctive navigation (see Section 3.4) with minimal additional processing time.

### 4.3 Tag Cloud View

The initial tag cloud shown in ConceptCloud includes tags from all attributes and objects in the context table (using the top concept in the lattice as the focus). This allows the user to select any tag from the extracted information. Tags in the initial tag cloud will be at their largest size and making selections in the initial tag cloud will result in tag clouds with smaller tags (cf. Figure 4.2), indicating that the cloud is only showing attribute tags from a subset of the total objects in the context table.

We provide functionality in the interface to remove unwanted categories from the tag cloud or limit clouds to one particular category. The cloud can also be adjusted to show only a certain number of tags or to show only tags that occur more than a given number of times. We also provide a search bar that allows users to start typing the name of a tag and then select the tag from a list of suggestions that are provided. In this way, if a user does initially know which tag they want to select, they are able to refine the view right away rather than needing to find the tag in the tag cloud.

#### 4.3.1 Display of Multiple Viewers

Customized visualizations can be created from the initial tag cloud by selecting relevant tags and by moving categories of tags into separate viewers. We use the JUnit [17] repository in order to illustrate how the viewers work in ConceptCloud. Initially all tags are shown in the same tag cloud (see Figure 4.4). The user can then dynamically separate different categories into different viewers. Figure 4.5 shows an example of viewers that have been created from the commit messages in the JUnit repository and the committer names in the JUnit repository. Users can keep constructing new viewers and the viewers are interlinked so that all viewers will be updated with each new selection or de-selection.

Viewers can also be opened with a “sticky” tag that always remains selected and cannot be deselected. This enables us to open multiple parallel viewers with different

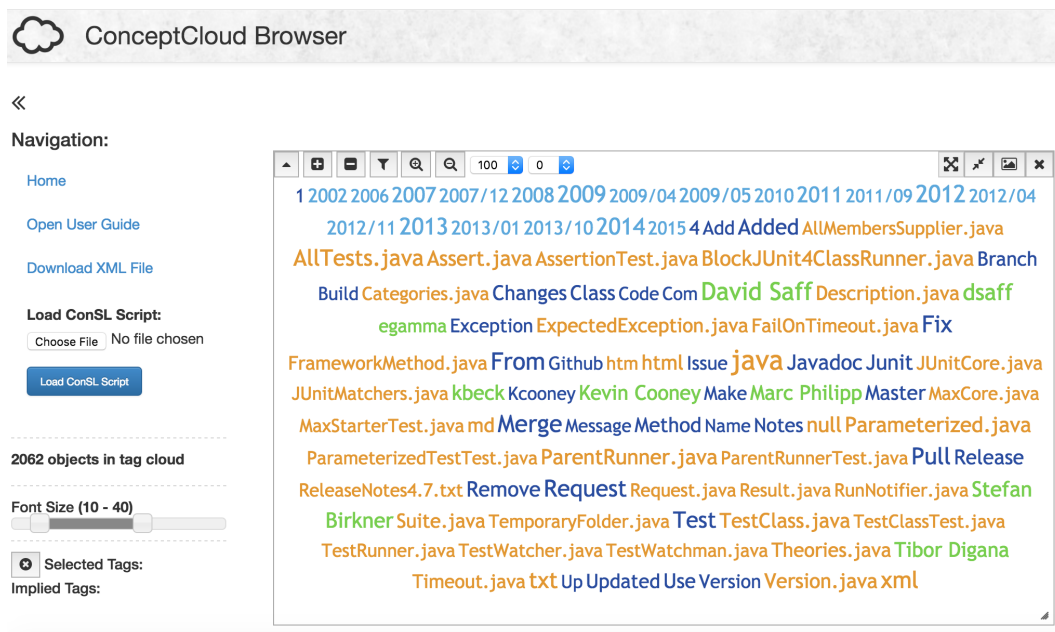


Figure 4.4: The initial tag cloud shown in ConceptCloud for the JUnit repository. The tag cloud is limited to showing the largest 100 tags. Initially all categories are shown in the same tag cloud and then these can be filtered and moved to separate tag clouds by the user.

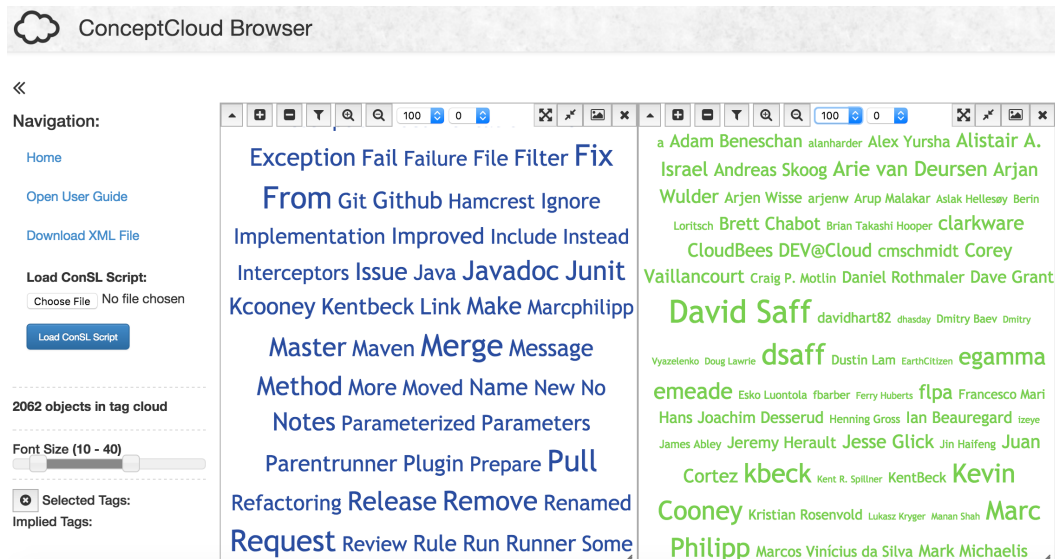


Figure 4.5: A multi-faceted tag cloud view shown in ConceptCloud. Words from commit messages are shown left in blue and author names are shown right in green. Both viewers are interlinked so that when a selection is made it will update all viewers.

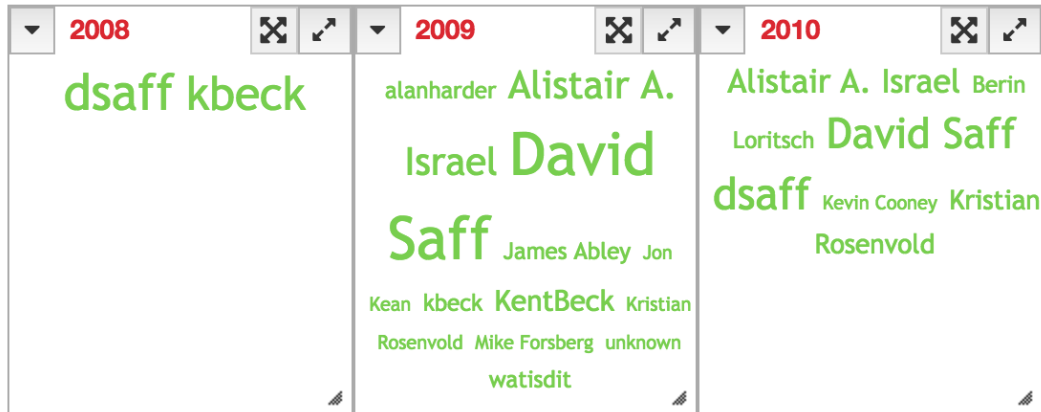


Figure 4.6: Example of a sticky tag viewer which allows us to show otherwise mutually exclusive views. 2008, 2009 and 2010 are selected as sticky tags. Sticky tags can never be de-selected from a sticky tag viewer but the viewer is updated with all other selections (but not other sticky tag selections) so that viewers remain interlinked

tag selections in the same category which update simultaneously when another tag is selected in any viewer. Sticky tags therefore enable us to show mutually exclusive views in multiple tag clouds next to each other and facilitate easy comparison of the views. Figure 4.6 shows an example of sticky tag viewers for the years 2008, 2009, and 2010 where the views are limited to showing only author tags. The sticky tags allow for easy comparison between the years. The viewers are still interlinked so they will be updated with all other selections. For example, if the user were to select a particular file tag in any view then the sticky tag viewers would be updated to show only which authors have worked on that file in 2008-2010.

### Handling Multiple Viewers in JavaScript

ConceptCloud allows users to construct multiple sticky tag or category viewers on the fly. Figure 4.7 illustrates how the viewers in ConceptCloud are controlled. Sticky tag viewers store their own selections, which are used in combination with the global selections, and category viewers only make use of the global selections. The class “WindowController” controls multiple “ViewControllers” which each in turn control multiple individual viewers. Each ViewController can have its own combination of selected tags (i.e., global selections and sticky tags) which are applied to each of its viewers.

#### 4.3.2 Viewer Customization

Each viewer provides various options for customization. Viewers can be restricted to showing only tags of a specific category, categories that have been previously

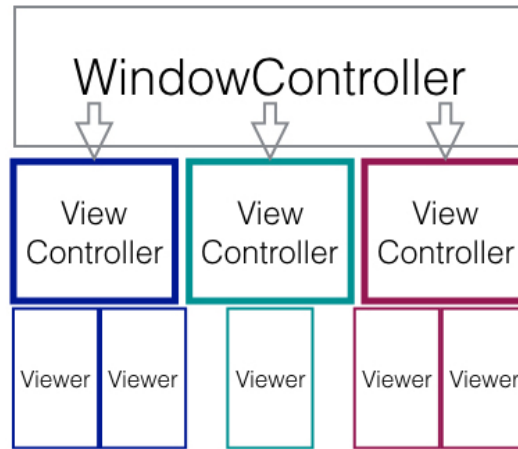


Figure 4.7: Hierarchy of JavaScript classes controlling different viewers in ConceptCloud. Each ViewController stores the current selections for its group of viewers. This can include both global selections (which are applicable for all ViewControllers) and sticky tag selections which are specific to an individual ViewController. The WindowController is responsible for managing all the ViewControllers and coordinating the global selections.

removed can be re-added to the viewer and additional categories can be removed. Buttons for zooming-in (i.e., increasing the tag size for all tags in the viewer) and zooming-out are provided individually for each viewer (see Figure 4.8(i-ii)). Figure 4.8(vi) shows the functionality that allows the number of tags in the cloud to be limited to showing only the largest X tags. Another option for limiting the tags in the cloud shows only tags whose occurrence count (used to calculate the tag size) is higher than a specific number (c.f. Figure 4.8(vii)).

ConceptCloud also provides an option to sort the current view according to descending order of tag count as opposed to the default alphabetic sorting (c.f. Figure 4.8(viii)). Each viewer can also be re-sized by the user and can be dragged into a new position to allow custom viewer configurations. Viewers that are no longer needed can be closed.

### 4.3.3 Table Viewer

A table viewer, which lists the information shown in the current tag cloud in a table format is also provided in ConceptCloud. The table shows only the information from objects for which the current tag selections apply. However, it is not affected by any viewer customizations (such as changing the number of tags or categories in the tag cloud). The table is scrollable and new rows are loaded dynamically as the user scrolls through the table. The table allows for additional information fields (such as the full title of an academic paper) to be shown on the interface, even when they do



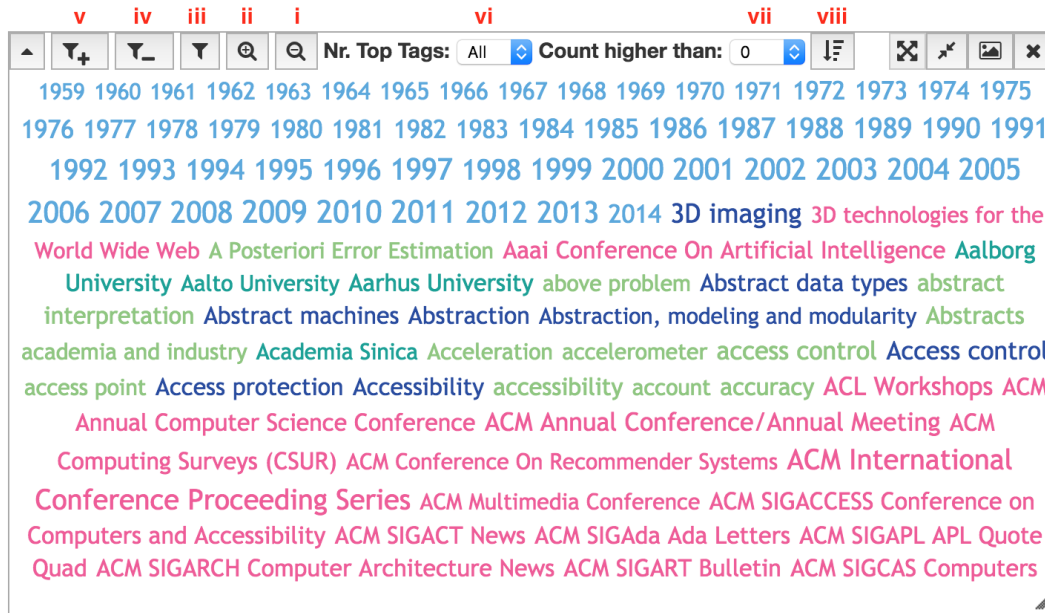


Figure 4.8: Available viewer customization options in ConceptCloud. (i) and (ii) show zoom out and zoom in functionalities respectively. (iii) shows the filtering option which allows the user to show only a specific category of tag in the cloud. (iv) and (v) allow users to add and remove individual categories of tags in the cloud. (vi) restricts the cloud to showing only the x largest tags and (vii) restricts the cloud to showing only tags whose count is higher than the selected value. (viii) sorts the tag cloud in descending order of the count of the tags.

not naturally make suitable tags for the tag cloud.

#### 4.4 ConSL: User Interface Scripting Language

We have developed a scripting language, ConSL, in order to construct and lay out multiple viewers in ConceptCloud simultaneously. Scripts can be written in order to open viewers for specific categories, open viewers with sticky tags and to customize the layout of the viewers in ConceptCloud's interface. While manually opening viewers from the ConceptCloud interface is useful for exploration of a dataset, opening multiple viewers (such as all sticky tags from a particular category) and manually laying them out can be time consuming. ConSL scripts provide a mechanism to easily recreate a particular viewer layout and can be used on multiple datasets so that the datasets can be compared using the same custom layout. The same script can also be loaded every time a dataset is loaded so that there is no need to manually configure the tag cloud layout on opening ConceptCloud. After executing a ConSL Script the user can still perform all available customizations through the interface.

ConSL scripts are compiled and used to generate JavaScript code that is executed

Listing 4.1: Example of a script written in ConSL for a version-control repository dataset. The author view shows only author tags and the for-loop opens an author view for each year tag selection. Views are sized at 50 percent of the full screen width and viewer menus are hidden.

```

define author_view as tag_view {
    category = 'author';
}
for tag_x in ['year'] {
    open author_view(tag_x);
}

layout author_view{
    width = '50%';
    menu = 'hidden';
}

```

in the browser where ConceptCloud is loaded. ConSL provides four main operations: defining a view, looping constructs, opening a view, and setting the layout. A view can be defined with one, multiple, or all categories of tags in the tag cloud. Views can then be opened with optional sticky tag selection arguments. The full grammar for ConSL is available in Appendix A.

For example, a view showing only the authors in a software repository can be defined and then this view can be opened with selection of year tag “2015”, to open a view showing all project authors in 2015. A for-loop construct is provided to open multiple viewers with sticky tags from all tags in a certain category, such as a sticky tag for each year (see Listing 4.1), which can be tedious to achieve manually through the interface. ConSL’s layout functionality allows the user to specify a precise layout and ordering for all the viewers. For example, Listing 4.1 shows a layout where each row will contain two viewers of equal width. The internal menus of each of these viewers will also be collapsed using a command in ConSL. Alternatively, using the interface’s drag and drop functionality to manually resize and layout multiple viewers can often lead to imprecise layouts. ConSL scripts can be loaded at the same time as a saved ConceptCloud context table, or from the tag cloud view. This enables users to load scripts after initial exploration of the dataset, when they have a better idea of what information they want to visualize.

## 4.5 Conclusion

We have implemented our approach in the ConceptCloud browser which can index data from XML and JSON files and allows users to browse the data through a web-interface. ConceptCloud is generic and can be used to explore various different kinds of semi-structured data. The ConSL scripting language allows users to write scripts

to determine the browser layout, so that different browser layouts can be saved and reloaded.

## Chapter 5

# Concept-Based Exploration of Version Control Archives

In this chapter we apply our approach to version control repositories and bug archives. We define multiple context types over data contained in Git and SVN repositories and make these available for exploration in the tag cloud interface. Software version control archives provide an interesting application for our approach as the individual commit-level data is very detailed but trends in the project are only observable when the commits are presented in an aggregate form. We evaluate our approach by performing four case studies, two of which are on open-source projects and two of which are on small industrial projects. We also conducted a user study with our undergraduate Software Engineering class in order to determine if ConceptCloud allowed untrained participants to answer questions about the history of a software project effectively.

### 5.1 Introduction

Version control repositories contain a wealth of implicit information that can be used to answer many questions about a project's development process, such as "Who worked on these files?", "Which developers collaborate?", "What are the co-changed methods?", or "What has happened in this project while I was away?". Answering such questions is a daily task for software developers [132]. Developers also rely on examining the history of a software project to keep up with changes, understand coding decisions and debug [52]. However, version control systems are not set up to make the necessary information easily accessible.

There are already many repository mining tools, such as Codebook [41], Hipikat [57], or the Information Fragments prototype [75], that use processed software repository data to show specific aspects of a project. However, these do not support ex-

ploratory search, and require the user to already have formulated a query in order to explore the dataset.

Developers find the ability to aggregate commits into groups and filter commits that apply to a certain topic useful [52]; this can be achieved through our interactive tag cloud interface. Our tag clouds can be used to visualize many aspects of the software project such as developer expertise (e.g., which developers have worked on particular files or directories and would be good candidates to ask questions about this functionality?), co-changed methods in a software project, project activity (e.g., which years and months has there been a lot of development, and on which parts of the system?) as well as developer collaboration (e.g., which developers are working together on which parts of the project?).

By using different objects in the formal contexts that are used to construct concept lattices, we are able to generate tag clouds that provide different perspectives on the same underlying data in the same familiar visualization. Existing repository visualizations implicitly hard-code the use of revisions as objects. In contrast, our foundation in formal concept analysis allows us to change the objects easily to get different insights on the repository and to generate visualizations that are distinctly not oriented towards time, distinguishing our approach from previous work.

## 5.2 Browsing Version Control Archives

### 5.2.1 Modeling Software Repositories

We use a simple repository model derived from [90] to formalize how we construct the contexts that underpin our browser: a *repository* is simply a collection of *versions* of a set of *files* that are grouped into *revisions*. Note that we follow the SVN terminology [115] here. In [90], versions are called revisions, while revisions are called modification requests; elsewhere revisions are called transactions.

A *version*  $v \in \mathcal{V}$  denotes the abstract state of a *file*  $f \in \mathcal{F}$  created by an *author*  $a \in \mathcal{A}$  at a *time*  $t \in \mathcal{T}$ . We ignore the actual file contents and only use meta-data and abstract modifications. Versions constitute a *version history* if they are ordered by a precedence relation  $\prec$  that holds only between versions of the same file and is compatible with the file creation times. We say that  $v$  *evolves into*  $v'$  if  $v \prec v'$  holds; two versions  $v_1$  and  $v_2$  are *merged* into  $v$  if  $v_1 \prec v$  and  $v_2 \prec v$ .

**Definition 12** Let  $\mathcal{V} \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{A}$  be a set of versions over files  $\mathcal{F}$  and  $\prec \subseteq \mathcal{V} \times \mathcal{V}$  be an irreflexive partial order.  $(\mathcal{V}, \prec)$  is called a version history iff  $v = (f, t, a) \in \mathcal{V}$ ,  $v' = (f', t', a') \in \mathcal{V}$ , and  $v \prec v'$  imply  $f = f'$  and  $t < t'$ .

A *revision*  $r$  is a set  $V$  of file versions that are committed to the *repository*  $\mathcal{R}$  at time  $t$  by an author  $a$ ; on commit, some meta-data (i.e., author, time, and an additional *log message*  $l \in \mathcal{L}$ ) is stored together with the versions. We assume that each revision  $r \in \mathcal{R}$  contains only one version of a file (which need not be the most recent version), and that each revision is uniquely determined by an abstract identifier  $id(r)$ .

**Definition 13** *Let  $(\mathcal{V}, \prec)$  be a version history and  $\mathcal{R} \subseteq \mathbb{P}(\mathcal{V}) \times \mathcal{T} \times \mathcal{A} \times \mathcal{L}$  be a set of revisions.  $\mathcal{R}$  is called a repository iff  $r = (V, t_r, a_r, l) \in \mathcal{R}$  and  $v = (f, t_v, a_v) \in V$  imply  $t_v \leq t_r$  and  $v \not\prec v'$  for all  $v' \in V$ .*

We can easily extend this basic model towards common revision control systems. For example, in CVS [144], the notions of versions and revisions are conflated; in our model we thus have for all revisions  $r = (V, t, a, l) \in \mathcal{R}$  that  $V = (f, t, a)$ . Note that we do not model revision tagging explicitly, but assume that the tags are part of the log messages. In SVN, each revision can only contain the most recent version of a file, and only the commit author and time are recorded but not the file author or modification time. Hence, in our model we thus have for all  $r = (V, t_r, a_r, l) \in \mathcal{R}$  and  $v = (f, t_f, a_f) \in \mathcal{V}$  that  $v \in V$  implies that  $t_f = t_r$  and  $a_f = a_r$ . Note that we are only interested in the linear sequence of revisions and therefore do not model explicit branching and merging, but again assume that this information is encoded into the log messages, if requested. For distributed revision control systems such as Git we analyze a clone of the repository. Note that clones of the repository in different states will generate different contexts, as the contexts are generated using the commit information extracted from the repository; therefore, if a repository is not up-to-date (i.e., has changes available to be pulled) then the generated context will differ from that of the up-to-date repository, as the list of commits differs.

### 5.3 Contexts from Repositories

In order to construct a concept lattice from repository data we need a context table. The first step in the construction of such a context table is to determine which field in the data will be taken as the object and which fields are suitable as attributes for that object. We use three different object types, namely revisions, files, and revision-file pairs (i.e., changes) in order to construct different types of contexts, which enables us to create different tag cloud visualizations for the same repository, providing new insights on the data. We are able to combine multiple data sources in the same context to support data fusion as object types in the context table need not be homogeneous. We use a combination of issue and version control data, in the same context, to provide a more complete overview of a project.

In the ConceptCloud browser, only one concept lattice and therefore context table can be explored at one time. Therefore, it is currently not possible to view data from different context types in the same window. However, the browser could be extended to present the different context types in different tabs in ConceptCloud so that users could explore them simultaneously.

### 5.3.1 Basic Preprocessing

When we construct context tables we pre-process the meta-data that we extract from the revision control system, in particular the log messages, file names, and commit times from each revision in the repository. We use a function  $\mathbb{W} : \mathcal{L} \rightarrow \mathbb{P}(\mathcal{W})$  that segments each log message into individual words  $w \in \mathcal{W}$ , removes words on a default stop list, and reduces each word to its stem, using the Apache Lucene implementation of Porter's [116] stemming algorithm.

We group both file names and commit times into increasingly coarser bins. For file names, we use a function  $\mathbb{D} : \mathcal{F} \rightarrow \mathbb{P}(\mathcal{F})$  that decomposes each file name into a set of all path prefixes, similar to recursively applying the Unix `dirname` command. For commit times, we use a function  $\mathbb{T} : \mathcal{T} \rightarrow \mathbb{P}(\mathcal{T})$  that truncates the times at different precision levels (days, months, and years).

In addition, we also use aggregators (such as aggregating files with the same names, even across directories) to capture regularities that appear across the bins, e.g., similarities between identically named files such as `README.txt` in different directories. We use  $d$ ,  $n$ , and  $t$ , respectively, to denote the mappings from each time to the corresponding weekday, and from each file to its base name and type, respectively.

Note that we do not perform more complicated pre-processing steps such as word sense disambiguation [111] or identity merging [125] as we prefer to leave the users in control of such aspects.

### 5.3.2 Revision-Based Contexts

In a revision-based context we interpret the *revisions*, represented by their *revision number*, as objects and the commit meta-data (e.g., author or words from the log message) as attributes; each revision is associated with its own meta-data as attribute. This context type represents the canonical view of repositories (see Definition 13). Its concepts are sets of revisions and their common attributes (e.g., all revisions that include a common set of files). Each commit induces a concept, since a developer can only commit one revision at any given time.

It is useful to get a historical overview of a project, for example to identify when the most changes have been made to a project, which developers have worked on particular files as well as which directories have been development hotspots.

**Definition 14** *Let  $\mathcal{R}$  be a repository, and  $\mathcal{A}_R = \mathcal{W} \cup \mathcal{A} \cup \mathcal{T} \cup \mathcal{F}$ . Then  $\mathcal{C}_R = (id(\mathcal{R}), \mathcal{A}_R, \mathcal{I}_R)$  is called the revision-based context of  $\mathcal{R}$  if for all  $r = (V, t, a, l) \in \mathcal{R}$ ,  $v = (f, t', a') \in V$ , and  $x \in \mathcal{A}_R$ , we have  $(r, x) \in \mathcal{I}_R$  iff*

- $x \in \mathbb{W}(l)$ , or
- $x = a$ , or
- $x = d(t)$  or  $x \in \mathbb{T}(t)$ , or
- $x = n(f)$  or  $x \in \mathbb{D}(f)$ , or
- $x = t(f)$ .

Definition 14 shows how a revision based context is constructed from a set of revisions where a revision is as part of a repository defined in Definition 13. The full set of attributes for the revision-based context is given by the union of all filenames, log messages, authors and times. The full set of objects in the revision-based context is given by the ID numbers for all revisions in the repository. An attribute will apply to a revision in the incidence relation if the attribute forms part of the words in the commit message, the author name, part of the full file path, part of the commit time or the file type.

### 5.3.3 File-Based Contexts

In a file-based context we interpret the *files* as objects but derive the attributes from the revisions' pre-processed meta-data; more precisely, each file receives all attributes from all revisions that involve the file. Concepts from such contexts are sets of files with common attributes (e.g., the set of all files on which a group of developers have all worked).

**Definition 15** *Let  $\mathcal{R}$  be a repository, and  $\mathcal{A}_F = \mathcal{W} \cup \mathcal{A} \cup \mathcal{T} \cup id(\mathcal{R})$ . Then  $\mathcal{C}_F = (\mathcal{F}, \mathcal{A}_F, \mathcal{I}_F)$  is called the file-based context of  $\mathcal{R}$  if for all  $r = (V, t, a, l) \in \mathcal{R}$ ,  $v = (f, t', a') \in V$ , and  $x \in \mathcal{A}_F$ , we have  $(f, x) \in \mathcal{I}_F$  iff*

- $x \in \mathbb{W}(l)$ , or
- $x = a$ , or
- $x = d(t)$  or  $x \in \mathbb{T}(t)$ , or
- $x = n(f)$  or  $x \in \mathbb{D}(f) \setminus \{f\}$ , or
- $x = t(f)$ , or



- $x = id(r)$ .

Definition 15 shows how a file-based context is constructed from a set of revisions where a revision is as part of a repository defined in Definition 13. The full set of attributes for the file-based context is given by the union of all revision IDs, log messages, authors and times. The full set of objects in the file-based context is given by the filenames for files changed in all revisions in the repository. An attribute will apply to a file in the incidence relation if the attribute forms part of the words in the commit message, the author name, part of the commit time, the file type or the revision ID.

Note that revision- and file-based contexts give complementary views on the repository. For example, the author tags from a revision-based context will be scaled according to the number of revisions that the author has committed over the project lifetime; during browsing only one author tag can be selected at a time since each revision has only one author. In a file-based context, the author tags will be scaled according to how many files a particular author has changed. Selecting an author tag will reveal all *collaborators*, i.e., all other authors who have also changed any of the same files. Selecting two author tags will then reveal the extent of their collaboration, i.e., all files they have both worked on. Therefore file-based contexts can be used to visualize the collaboration in the project, showing which developers work together and on which files.

#### 5.3.4 Change-Based Contexts

In a change-based context we use *pairs of files and revisions* as objects, so that for example (`hello.java`, `revision-1`) and (`hello.java`, `revision-3`) become separate objects in the context. This allows us to use the *content* of the files as additional attributes, which we cannot do with revision- or file-based contexts. In our implementation we focus on the changes (rather than the entire contents), and use a lightweight fact extractor [110] to get the signatures of the changed methods from each file. We could therefore have, for example the attributes `public int equals()`, `public static void main()`, and Alice associated with the object (`hello.java`, `revision-1`) to represent the fact that `revision-1` by Alice changes the methods `equals()` and `main()`. Selecting a method tag  $m$  then produces a tag cloud which contains all other methods that have been *co-changed* with  $m$ , scaled according to how often they have been changed together (cf. Figure 5.1). Therefore change-based contexts can be used to construct visualizations that depict the co-changed methods in the project as well as showing other method information, for example, which methods are development hotspots and in which time periods.

In our model we assume a set  $\mathcal{M}$  of abstract *modifications* (in the spirit of the atomic changes of Ren et al. [122]), and use  $\Delta(v', v) \subseteq \mathcal{M}$  to denote the (non-symmetric) difference between two versions  $v' \prec v$  of a file.

**Definition 16** *Let  $\mathcal{R}$  be a repository, and  $\mathcal{A}_C = \mathcal{W} \cup \mathcal{A} \cup \mathcal{T} \cup \mathcal{F} \cup id(\mathcal{R}) \cup \mathcal{M}$ . Then  $\mathcal{C}_C = (\mathcal{F} \times id(\mathcal{R}), \mathcal{A}_C, \mathcal{I}_C)$  is called the change-based context of  $\mathcal{R}$  if for all  $r = (V, t, a, l) \in \mathcal{R}$ ,  $v = (f, t', a') \in V$ ,  $v' \in V$  with  $v' \prec v$ , and  $x \in \mathcal{A}_C$ , we have  $((f, r), x) \in \mathcal{I}_C$  iff*

- $x \in \mathbb{W}(l)$ , or
- $x = a$ , or
- $x = d(t)$  or  $x \in \mathbb{T}(t)$ , or
- $x = n(f)$  or  $x \in \mathbb{D}(f)$ , or
- $x = id(r)$ , or
- $x \in \Delta(v', v)$ .

Definition 16 shows how a change-based context is constructed from a set of revisions where a revision is as part of a repository defined in Definition 13. The full set of attributes for the change-based context is given by the union of all revision IDs, log messages, authors, filenames and times. The full set of objects in the change-based context is given by pairs of filenames for files changed and revision IDs for all revisions in the repository. An attribute will apply to a change (filename and revision pair) in the incidence relation if the attribute forms part of the words in the commit message, the author name, part of the commit time, the file type, filename, the revision ID or a change in the file that occurred between two revisions.

### 5.3.5 Combined Contexts: Bug Reports and Revision Control

#### Data

Software development projects often make use of dedicated tools for different tasks, such as issue databases, task trackers, and source code repositories, or use a tool that provides a combination of these such as GitHub [8]. Moreover, archive entries can be linked across the different tools, by for example, adding an issue identifier to the log message of a revision which references that issue. Ideally, visualization tools should be able to “fuse” the information from different archives for the same project into a single combined data structure, such as Hipikat’s uniform artifact database [57] or Codebook’s central graph [41].

Here, we combine data from multiple archives (or different features of GitHub) into a single context using multiple object types. In particular, we combine repository

data and GitHub issue data in the same context. In the combined contexts we use the revisions and bug reports as objects (since the object types in the context table need not be homogeneous) and derive the attributes from both the revisions' pre-processed meta-data and text from the bug reports. Therefore, where bug reports and revisions share a common attribute they will be grouped together in the same concepts, indicating the relation of the bug reports to the revisions. The combined context gives a more complete overview of the project activities.

Note that the objects in a combined context are a disjoint *union* of revisions and issues; this is different to the construction of the change-based contexts where the objects are *pairs* of revisions and files. The combined context's attributes are the union of the original attributes for both the revisions and issues and each object keeps its own attributes. We merge corresponding attribute categories from the data sources, e.g., log messages and issue descriptions. This assumes that words have the same meaning in the different archives, but in return it provides us with implicit links between bugs and revisions that both talk about a specific topic (e.g., "Linux"), because their log messages and descriptions share a common attribute. The issues and revisions are therefore connected automatically, without the need to create any links, as for example described in [135]. However, for a data source such as GitHub, which stores explicit references between commits and issues, we are able to link these in the context table by using a "surrogate key" attribute which we assign to both the revision object and the issue object in the context table.

Note that the surrogate key does not become the object type as each revision is not necessarily linked to a particular issue and vice versa. The revision object and the issue object are linked in the concept lattice because they both share a common attribute (the surrogate key) which is unique to the individual issue and revision and functions as a surrogate key. A surrogate key is therefore an additional attribute assigned to both an issue and a revision which serves exclusively to indicate an explicit link between the revision and the issue in the concept lattice.

Section 5.5.2 provides examples of tag clouds generated from Git repositories and issues in the GitHub issue-tracking system. Combined contexts can be used to visualize which files have been changed when a bug has been fixed as well as showing the project activity both in terms of commits and issue reports.

## 5.4 Advanced Customizations in ConceptCloud

We have extended ConceptCloud to support a number of advanced visualizations that enable customized views for repository browsing. We show how the context table can be constructed only for a specific range of commits (a subset of the full

object set) and how the attributes categories can be customized to encode additional information.

### 5.4.1 Personalization in Tag Clouds

We can personalize a tag cloud for a particular developer by identifying all tags that apply to that developer (e.g., files they have changed) in our pre-processing step. We then assign these tags to different categories than the tags from the remaining commits (such as “file of interest”), and render them in a different color. In the personalized tag cloud, the files that have been changed by that particular developer will thus be easily identifiable in views even when the tag for that developer has not been selected.

### 5.4.2 Filtering Tag Clouds

If we want to analyze only a particular section of a repository (e.g., only the portion since we started working on the project) we can restrict the revision range from which the context table is constructed. Our pre-processing offers different ways of specifying the ranges of interest, such as processing only a certain number of commits or processing only commits falling between a specified start and end date.

### 5.4.3 Customized Visualizations

One example of a customized visualization that can be created in ConceptCloud uses the personalization and filtering techniques to form a *vacation cloud*. The vacation cloud allows ConceptCloud to highlight answers to the question “What happened in my project while I was away?” with a vacation cloud as for example shown in Figure 5.1. This is constructed from a change-based context where file and method tags have been personalized to the developer (here David Saff) and revisions have been filtered by the date of his last commit.

The initial tag cloud shows in which revision most files were changed (1856), when most changes happened (2014/06/18), or which developers have made most changes (Alex Yursha and Kevin Cooney, cf. Figure 5.1). Tag colors indicate the corresponding categories and selected tags are shown in red. The words from the commit messages indicate that most changes were either pull requests or stylistic in nature, as indicated by prominent tags such as **Change**, **Codingstyle**, **Legacycodingstyle**, or **Remove**. However, the overall view of the changes in Figure 5.1 does not provide us with many detailed insights into the data and we refine the view by selecting tags in order to discover more. Selecting a developer gives a more detailed view of their changes and selecting one of the most active developers, Yursha, reveals

that he has only committed one revision that contains stylistic changes to many files. Alternatively, selecting Cooney reveals that he has merged in several pull requests (cf. Figure 5.1) which contain changes to files that Saff has previously worked on (such as `AllTests.java`).

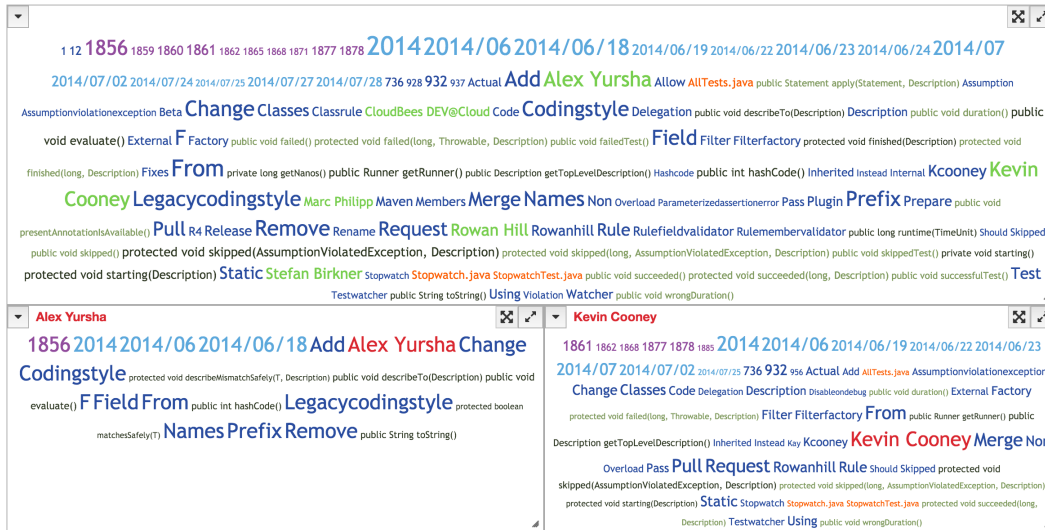


Figure 5.1: JUnit: vacation cloud for David Saff constructed from a change-based context. Main tag cloud view (top). Changes by Alex Yursha (bottom left) and Kevin Cooney (bottom right). Alex Yursha and Kevin Cooney are selected with sticky tags. Only tags with occurrence greater than two are shown.

Selecting further tags (e.g., `From` and `Rowanhill`) brings out further details (e.g., about the pull requests from `Rowanhill`). The cloud also shows how often files and methods have been changed; it uses different colors to distinguish changes in files previously changed by David Saff from those in other files. We can therefore see that different variants of the method “skipped” were a development hotspot during Saff’s absence; we can further see that variants with different signatures were added (shown in light grey), on top of the changes to the variants that Saff has also worked on (shown in dark grey).

## 5.5 Illustrative Case Studies

We performed case studies on different repositories to show what types of insights can be gathered from the repositories using `ConceptCloud`. The JUnit case study (see Section 5.5.1) allowed us to draw comparisons to a previous case study performed by Weissgerber et al. [140, 148]. The RubyGems case study (see Section 5.5.2) presents a combined context of both revisions and issues from the GitHub issue tracking feature. We also performed two industrial case studies with two different teams in

Sections 5.5.3 and 5.5.4 and verified our observations with the project managers. The industrial case studies allowed us to verify whether the insights that we gather using ConceptCloud are appropriate.

### 5.5.1 JUnit Repository

JUnit is a popular open-source testing framework for Java which has been used in previous case studies [140, 148]. Here we repeated Weissgerber’s case study [148], which investigates developer roles up until 2006, and extend it to a more current date. We show that we can easily extend the previous observations on the repository through our interface even though our interface was not specialized only to identify collaboration patterns. We created the revision-based context for the JUnit project from its first revision in 03/12/2000 until 26/02/2014 (1772 revisions).

#### 5.5.1.1 Overview

In order to get an initial view of the project we opened a commit time view and restricted it to years. This showed that project activity increases dramatically from the first full year in 2001 until 2007 and remains relatively steady thereafter. Selecting the year tag 2000 in the full cloud showed us that developer `egamma` started the project in December 2000. In an author cloud for the first full year of development (2001) we see that developers `kbeck` and `emeade` joined the project in 2001 but `egamma` remained the most prolific author in that year (cf. [148]).

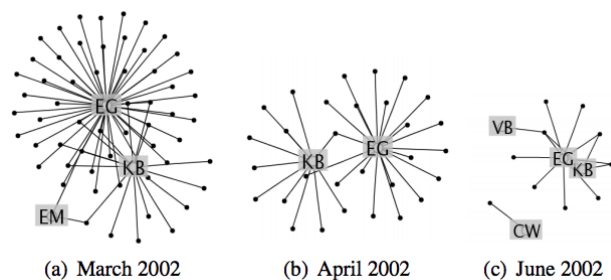


Figure 5.2: The Author File Graph from a section of the JUnit repository as shown in [148]. The Author File Graph shows author nodes linked to files that they have worked on which shows the overall activity of an author as well as the extent of collaboration between authors.

#### 5.5.1.2 Authors by Month

Weissgerber et al. [148] look specifically at the file changes made in the months March to June 2002. To repeat this we opened viewers with “sticky tags” for March, April,

and June 2002 (there was no commit in May 2002) and limited these to show only author and file tags (cf. Figure 5.3, top). Selecting an author tag shows us which files the author has worked on in each month. Figure 5.3 shows kbeck's contribution reduced in the given period. The cloud for June 2002 showed the addition of developers vbossica and clarkware to the project.

Selecting the file `TestRunner.java` showed that egamma and kbeck had changed this file in April 2002 and only egamma had made changes in June 2002 (cf. Figure 5.3 (bottom)). We also see that there are a group of files which have been changed at the same time.

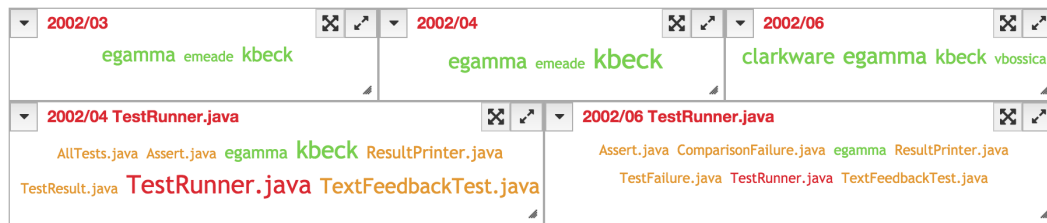


Figure 5.3: JUnit: author clouds (top); changes to `TestRunner.java` (bottom). Tag clouds constructed from a file-based context and months/files are selected as sticky tags.

### 5.5.1.3 Conclusions

ConceptCloud allowed us to gather the same insights related to author activity and collaboration as the dedicated tool presented in [148]. However, in contrast to [148], it does not produce a static picture but allows the user to refine the analysis, and access the other information (e.g., log messages) that remains available.

## 5.5.2 RubyGems Repository

We constructed the combined context for commits and issues from the RubyGems GitHub repository [28] to show how we can combine issue and repository data in the same tag cloud. The GitHub issue tracking system provides links between issues and commits that either close an issue or reference it. We extract these links, using the GitHub API, to create explicit links between issues and commits in our tag cloud, but we also extract keywords from the issues and commit messages and use these to create implicit links between issues and commits that discuss the same topics. For other issue tracking systems that do not include explicit links between issues and commits we would still be able to extract implicit links from the commit messages.

Linked issues and commits appear in the same tag cloud showing which files have been changed in order to close an issue. For example, Figure 5.4 shows the tag

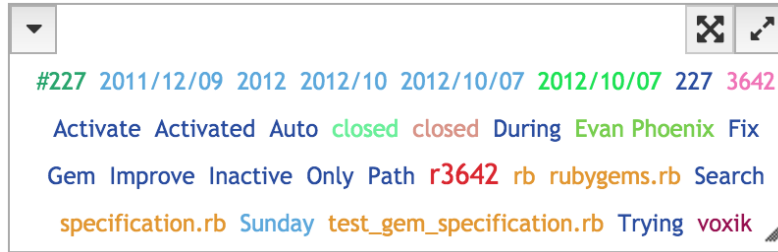


Figure 5.4: RubyGems: Tag cloud for commit 3642 which closes issue 227 in RubyGems

cloud containing information for issue 227 which was closed by commit 3642. We can immediately see that files `rubygems.rb` and `specification.rb` were fixed in relation to the bug reported about inactive gems. We see here tags `#227` as well as tag `227`, where `#227` represents the issue object and `227` is part of the commit message for commit 3642. We see also tags `r3642` and `3642`, where `3642` represents the revision object and `r3642` is used as a link between both the revision and the issue objects. Therefore, while `3642` and `#227` occur only once, `r3642` occurs twice (and appears bigger) in the tag cloud as it is an attribute which applies to both the issue and the revision objects in the tag cloud (i.e., `r3642` is the surrogate key that connects the issue and revision).

Additionally, we can explore all commits and issues that discuss a particular topic such as “gem install”. Figure 5.5 shows the main files (orange tags), committers (green tags) and GitHub issue reporters (maroon tags) that are associated with the keywords `gem` and `install`. We can further restrict the cloud to showing only commits that have closed a bug report by selecting the bug report status `closed` (see Figure 5.6). We see that Eric Hodel is the only author who makes commits closing issues that mention `gem install` and these commits only occur in 2013 and 2014. This indicates that while other authors have also made commits on the topic, Eric Hodel is likely responsible for this area as he has either fixed issues referring to `gem install` or has been responsible for closing these issues when merging a pull request from another developer.

### 5.5.2.1 Conclusions

ConceptCloud can be used successfully to combine multiple data sources to get more detailed information on a specific project. We can fuse issue and repository data into the same underlying context table and explore commits that are related to specific issues in the tag cloud interface.



Listing 5.1: ConSL Script for the View in Figure 5.5

```

define main as tag_view {
  category = 'message';
}
define files_view as tag_view {
  category = 'filename';
}
define committers as tag_view {
  category = 'author';
}
define reporters as tag_view {
  category = 'issue_reporter';
}

open main('Gem, Install');
open files_view('Gem, Install');
open committers('Gem, Install');
open reporters('Gem, Install');

layout main {
  views_in_a_row = 1;
  menu = 'hidden';
}
layout files_view {
  views_in_a_row = 2; menu = 'hidden';
}
layout committers {
  views_in_a_row = 0.5;
  width = '15%'; menu = 'hidden';
}
layout reporters {
  views_in_a_row = 0.5; menu = 'hidden';
}

```

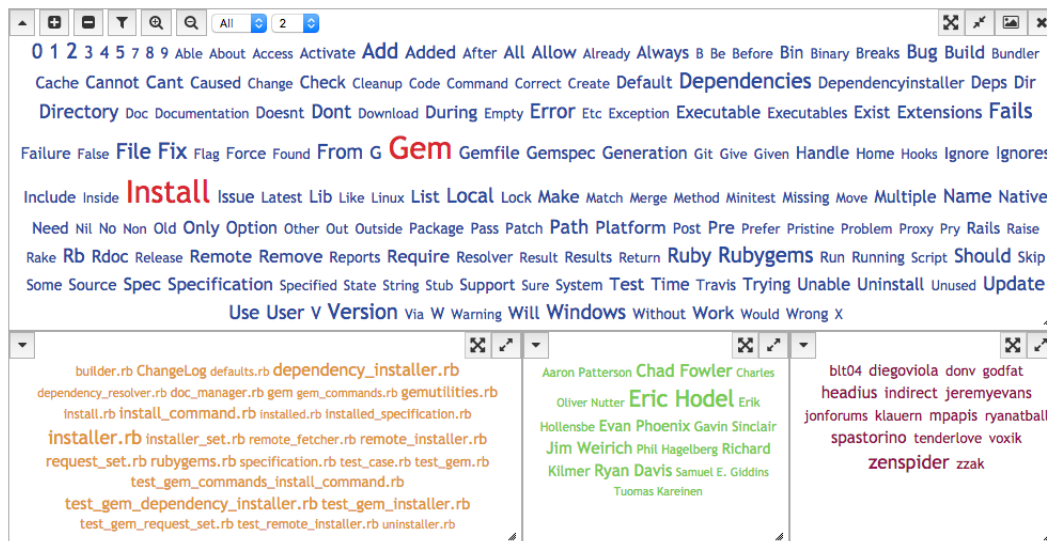


Figure 5.5: RubyGems: Main changed files, committers and bug reporters from commits and issues mentioning Gem Install. Tag clouds constructed from a combined context of GitHub issues and commits.

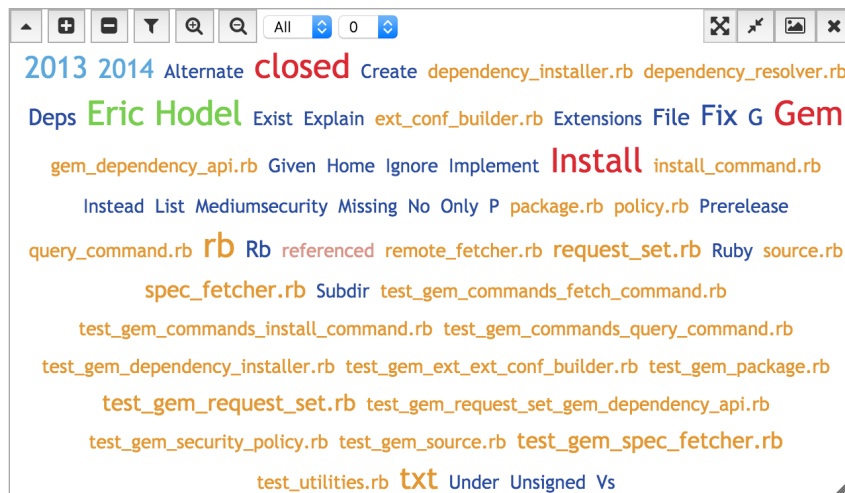


Figure 5.6: RubyGems: Changed files, committers from commits closing issues mentioning Gem Install.

### 5.5.3 Industrial Case Study: Ruby on Rails Web Application

We analyzed the Git repository of a small commercial development project which implements in Ruby on Rails a crowd-funding website that handles donations for solar lights. The repository contains 490 revisions in 219 files with approximately 8900 lines of code that were developed over a period of two months by seven developers.

The goal of this case study was to see whether the ConceptCloud browser can be used to quickly gain an understanding of an unknown project's organizational structure. We therefore created a revision-based context, browsed the tag cloud for approximately two hours and then checked our observations with the project manager who confirmed them.

#### 5.5.3.1 Project Activity

The commit time view (cf. Figure 5.7) shows that the project had two major activity spikes (2014/1/14-15 and 2014/1/29-30). The project manager confirmed that these time periods coincided with project demonstrations. However, when we investigated the log message tags we were unable to see any demonstration references in the log messages themselves. The second spike is trailed by a number of commits on Saturday 2014/2/1, by developer PP who works mostly on Saturdays (cf. Figure. 5.8). The project manager confirmed that PP was working on contract and mostly worked on the weekends.



Figure 5.7: Tag Cloud showing all dates of all commits to the project where there are noticeable activity spikes from 2014/1/14-15 and 2014/1/29-30

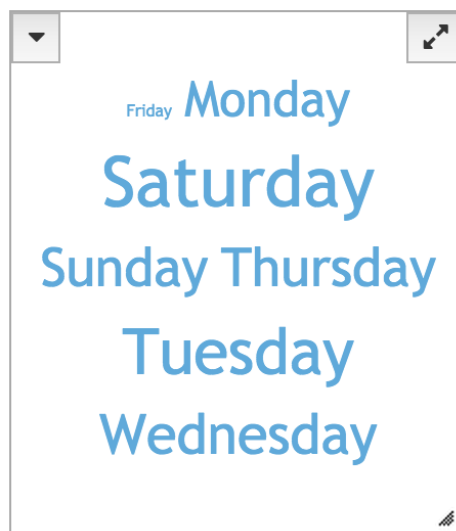


Figure 5.8: Weekdays of developer PP showing that he makes most commits on Saturdays and works less during the week.

### 5.5.3.2 Developer Activity

The author view (cf. Figure 5.9a) shows that the project involves seven developers, with three developers (CK, DR, PP) evenly sharing the main work load (80% of the revisions) on the project. Selecting the respective names of two developers (AE, GM)

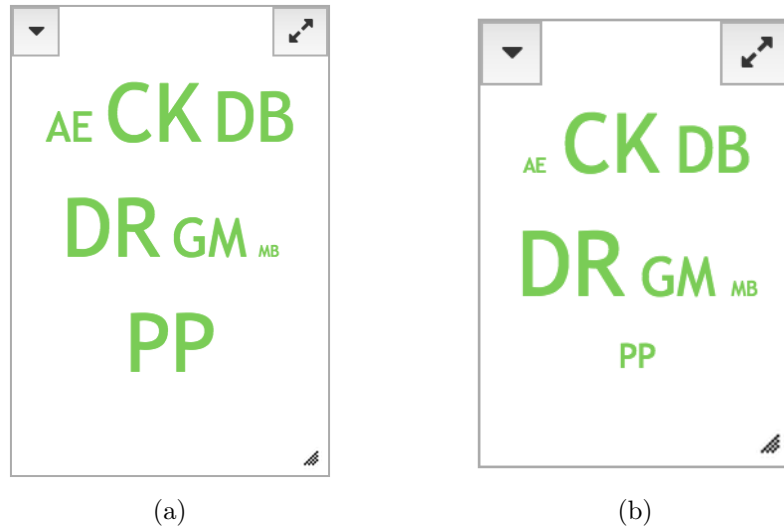


Figure 5.9: Tag Cloud showing (a) All developers working on the project sized according to the number of commits they have been making (b) Developers of merged pull request sized according to the number of commits they made where merged pull request appeared in the log message text.

reveals (in the commit time view) that these were only active towards the end of the project.

### 5.5.3.3 Developer Expertise

Selecting the keywords `merged`, `pull` and `request` in the main cloud shows (in the author view, cf. Figure 5.9b) that most Git pull requests were merged by DR, followed by CK. This indicates that these two developers are the main architects of the system. Selecting DR and opening a directory view (cf. Figure 5.10) shows that DR is mostly merging requests concerning files in the `app/controllers`, `app/models`, and `app/views` directories and is thus responsible for the system’s functionality, while CK works mostly in the `app/views` and `app/assets/stylesheets` directory and is thus responsible for the system’s appearance.

However, deselecting the keywords and just selecting the `app/assets/stylesheets` directory indicates that PP is actually implementing most of the visual functionality.

Selecting individual keywords (e.g., `video` or `facebook`) shows that the related parts of the system functionality were implemented by one of the lead developers, often with the support of a second developer to integrate the functionality into the web pages (e.g., `video` by PP and GM, cf. Fig. 5.11).

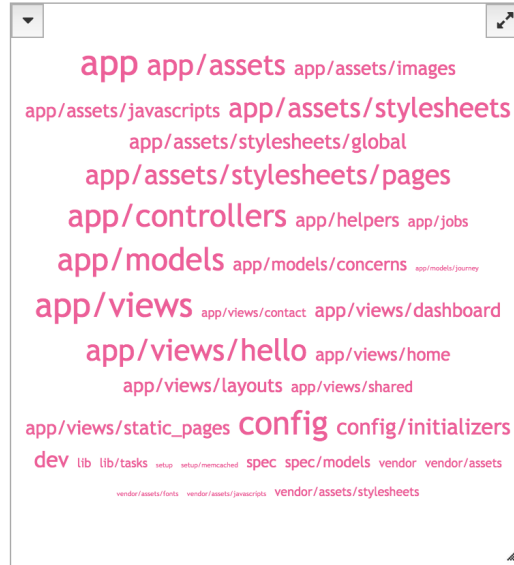


Figure 5.10: Directory cloud of developer DR which indicates that he works on `app/controllers`, `app/models`, and `app/views` directories and is responsible for the system functionality.

#### 5.5.3.4 Conclusions

We were able to extract the team structure and developer responsibilities from the repository which would be helpful for new team members. We confirmed all our observations with the project manager who was surprised by the insights that we were able to gather using the commit history. The category-specific tag clouds were instrumental in gaining insight about certain aspects of the project (e.g., developers) and finding this information much more quickly. The multi-faceted visualization allowed us to analyze and explore different aspects of the information concurrently, which made observations more obvious. The full tag cloud containing information from all categories gives us a complementary unified view from which we can re-direct our exploration after a navigation step.

Overall, our insights come from the incremental nature of the exploration: each step reveals a new view into the repository and each view in turn suggests the next steps, through its most prominent tags.

#### 5.5.4 Industrial Case Study: Mobile Application

We performed a further case study on the Git repository of a small, local non-profit organization. This project develops an educational service comprising of a mobile app, backend and data analytics. We analyzed the project from its start (08/2015) up until the app's release to the Google Play Store (01/2016). Note that we use

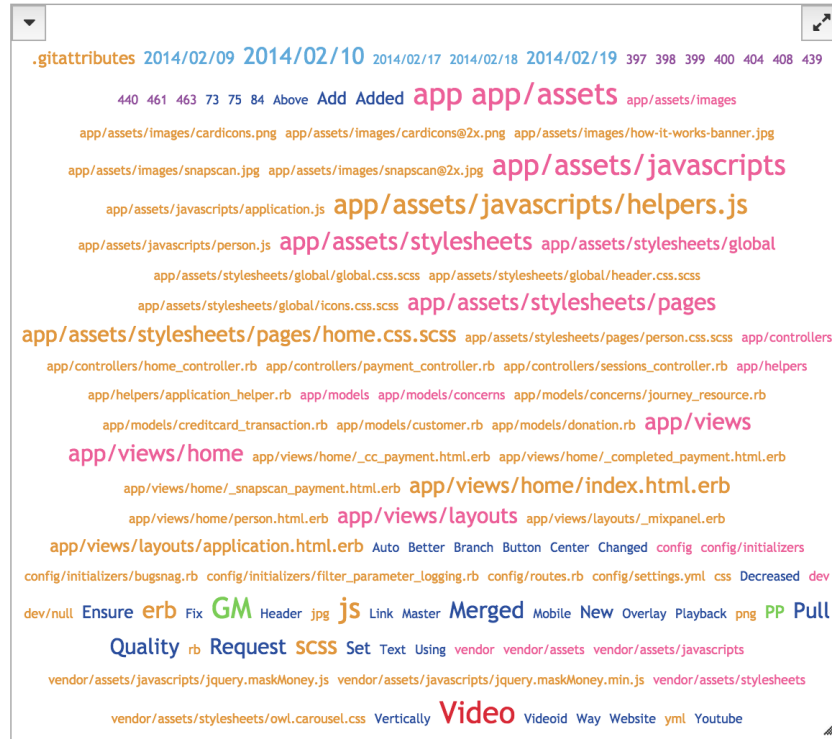


Figure 5.11: Cloud after selection of tag Video. This cloud shows all authors who made commits mentioning the video functionality, as well as the dates on which these commits were made and the files and directories that were touched as part of these commits.

only abbreviations of these developers' commit names in this study to preserve their anonymity.

The goal of this case study was to see whether insights that be gathered by using the ConceptCloud browser are appropriate. Note that while the insights gathered may not be surprising to the project manager of a small localized development team, the type of information that we have gathered from the repository in this study would be useful for large distributed teams. We confirmed all our insights with the project manager.

#### 5.5.4.1 Project Contributors

By creating author viewers for each month from the revision-based context of the project (Figure 5.12) we can see that the project started towards the end of August 2015 with only three developers. In September all three developers contributed in similar amounts to the project. In October two more developers (CM and P9) joined and overall commit activity of the developers greatly increased. Additional contributors RS and S also joined the project in November, and this team remained

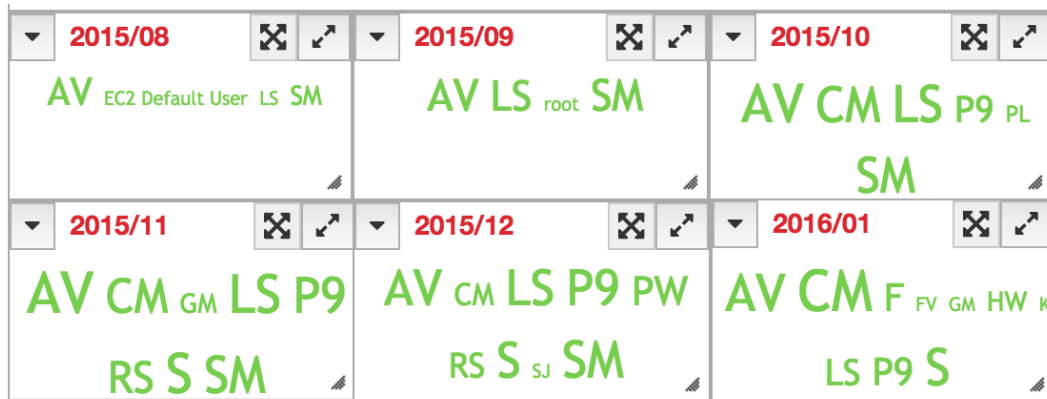


Figure 5.12: Developer contributors over project from project start to first release. Tag clouds generated from a revision-based context. Months are selected as sticky tags.

relatively stable with only the addition of PW in December. The team structure changed again in January with SM, PW and RS leaving but two new (and therefore less-active) contributors, HW and F joining the project. In each month the developers (excluding the new additions) appear to be sharing the workload uniformly. We see that the project was expanding but also that there was a high developer churn. The project manager confirmed that contractor SM was replaced by two new full time developers in January.

We can also observe other infrequent contributors (1-3 commits, which show up as small tags in the tag cloud) which on further investigation appear to be alternative aliases (particularly GitHub usernames) for some of the contributors (i.e., such as a developer editing the ReadMe file directly on GitHub and the commit being recorded with their GitHub username). These alias characteristics could also be incorporated to identity merging techniques as identity merging in projects is a difficult problem [83].

#### 5.5.4.2 Developer Collaboration

By creating the file-based context of the project we can observe collaborations between the developers. Selecting developer LS (Figure 5.13) shows that he often collaborates with developers SM and P9. However, there are a small number of files common to other team members as well. Developers F and HW have also begun collaborating with LS since they have joined the project. If we select an additional tag for developer AV (who shows up only as a small tag) and show which files both AV and LS have changed, we see that the `gitignore` file is the only file that is common to most of the development team, indicating that many of the developers have distinct roles within the team.



Figure 5.13: Collaboration with developer LS observed using a tag cloud built from a file-based context. The size of the tags of other contributors indicate how many files they have worked on that LS has also worked on.

If we select the tags for LS’s collaborators SM and P9 and show the tag cloud for the changed directories (Figure 5.14a), we see a directory structure that indicates that these developers work on the Android client. This cloud also confirms that developers F and HW have also begun working on the Android client.

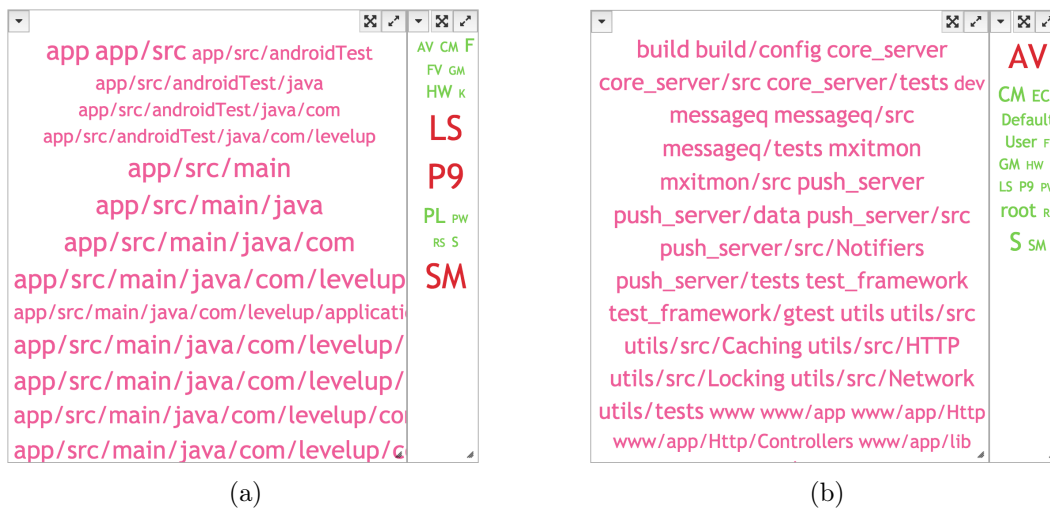


Figure 5.14: Directories and collaboration of developers (a) LS, SM and P9 and (b) AV. Collaboration is observed from the tag cloud built from a file-based context.

The directory cloud of developer AV (Figure 5.14b), who collaborates mostly with S and CM shows a very different directory structure appearing to be concerned with backend development. Therefore, we see a clear separation of responsibilities among the development team. However, when we investigate the collaboration clouds of S and CM individually we see that these three developers each work on a number of files that are not touched by other team members. If one of these developers were to leave the team there would be a large number of files that no other team member would be familiar with. Therefore, we see that the backend team has a low “bus factor” [3].

Contributer RS does not appear as one of the main collaborators of AV’s team (backend development) or LS’s team (android application) and on further investiga-



tion of RS's changed directories we see that he is mostly contributing images to the project.

### 5.5.4.3 Commit Activity

Comparing the revision-based and file-based views on the weekdays on which the developers have been making commits (Figure 5.15) we see that the most commit activity occurs between Tuesdays and Thursdays with there being less activity on Mondays and Fridays and very little over the weekends (Figure 5.15 (left)). This is consistent with what we expect from a full-time commercial development team.

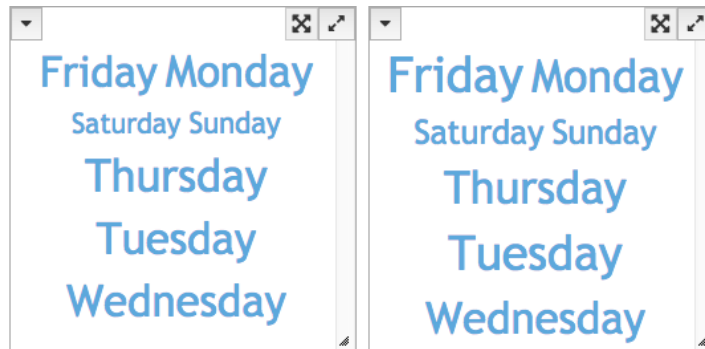


Figure 5.15: Weekdays of developer commits with tags sized according to number of commits (left) and tags sized according to number of files changed (right).

Observing the number of files changed on each weekday (Figure 5.15 (right and left)) shows that the tag for Friday is slightly larger in the file-based view than in the revision-based view which indicates that fewer commits are made on Fridays that generally touch more files. This would be consistent with developers storing their changes before the weekend in fewer but larger commits. The project manager also indicated that bi-weekly sprint planning takes place on a Friday, which could also explain the fewer but larger commits observed on Fridays.

### 5.5.4.4 Commit Messages

Examining the most frequent words used in commit messages in the first full month of the project (09/2015) and comparing those to the commit messages in 01/2016 (Figure 5.16) we see that the initial activity was largely concerned with Facebook integration. In the last month examined (01/2016) we see that the activity is more centered around bug fixes and user interface changes (Images, Styling).

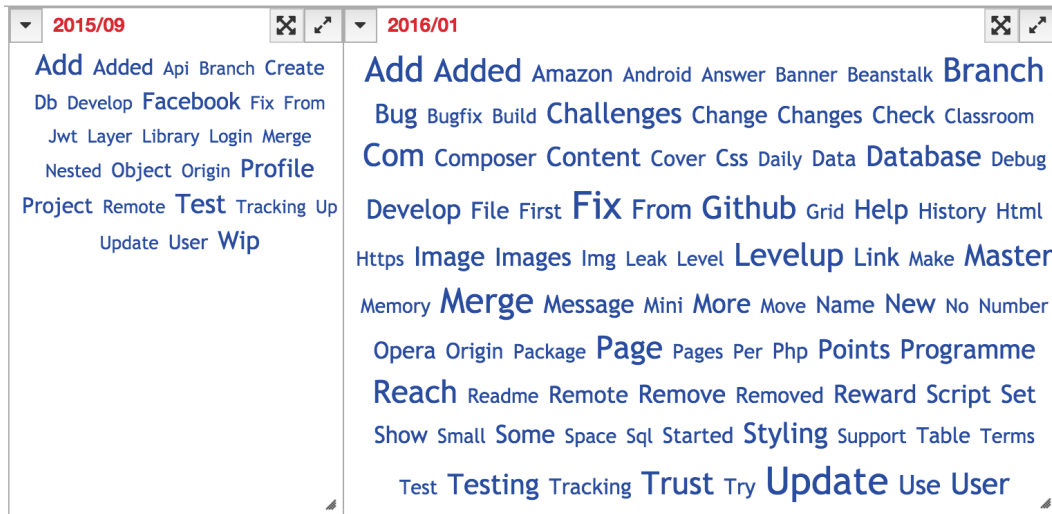


Figure 5.16: Popular keywords from commit messages in the first and last month analyzed.

### 5.5.5 Conclusions

Using ConceptCloud we are able to get an overview of the collaboration and work patterns in both open source and industrial projects. We see that the different context types give us different views on the project (collaboration vs. commit activity) and that additional project information such as the issue-reports can be merged into the same context to provide more detailed information on a project.

The information that we can gather using ConceptCloud was deemed appropriate by the project managers of two different industrial projects. We were able to use ConceptCloud to quickly get an overview of the project's history and the collaboration within the teams.

Comparing our observations from an industrial project to those made from open-source projects we observe that the commit activity in the industrial project is much more regular and the contributions are shared relatively evenly among the contributors. The development team of the commercial project is also separated into smaller groups, of approximately three, that work consistently on one aspect of the project. However in the open-source projects we observe one main contributor who has a much higher activity than the others during their involvement; when this contributor leaves the project another developer takes over this role.

## 5.6 Performance Evaluation

We used ConceptCloud on a medium-sized server (64GB RAM, 2 Xeon 8-core 2.0Ghz CPUs) to analyze several Git and SVN repositories in order to evaluate its perfor-

mance. FCA is commonly associated with high run-times and so we evaluate ConceptCloud’s performance on a variety of repositories to illustrate the feasibility of our approach.

Table 5.1: Performance metrics for revision-based contexts

Project	Type	$ O $	$ A $	$ I $	Indexing time (s)	Initial cloud drawing time (s)
Subversive <sup>1</sup>	SVN	1,511	8,222	88,090	55.5	1.8
JUnit <sup>2</sup>	Git	1,905	5,959	66,242	8.0	1.9
AngularJS <sup>3</sup>	Git	5,547	9,055	133,436	116.2	2.8
Spring <sup>4</sup>	Git	9,017	40,332	540,813	43.4	14.8
Valgrind <sup>5</sup>	SVN	10,989	29,009	348,136	176.6	40.0
Django <sup>6</sup>	Git	18,471	38,821	583,701	58.4	11.0
Moodle <sup>7</sup>	Git	69,550	154,834	2,222,486	333.2	45.7
DPorts <sup>8</sup>	Git	155,627	196,850	2,917,269	2,049.9	892.8

We created revision-based contexts (using local clones of Git repositories and remotely accessing the SVN repositories). Table 5.1 summarizes the characteristics of and runtimes for these repositories, showing the number of revisions  $|O|$ , the number of attributes  $|A|$ , and the size of the incidence relation (i.e., the number of object/attribute pairs)  $|I|$ , as well as the time to create the context table (i.e., indexing) and to draw the repository’s full tag cloud.

We see that the indexing times (including the extraction of all of the log information for the repositories) are only a few seconds for smaller repositories, and a few minutes for medium-sized ones; even the largest repository with 155627 revisions requires only 34 minutes. Note that these times are not directly related to either the size or the density (i.e.,  $|I|$ ) of the context tables but are to a large extent determined by the (lexical) pre-processing.

<sup>1</sup><https://dev.eclipse.org/svnroot/technology/org.eclipse.subversive/>. Analyzed December 2006 to September 2014.

<sup>2</sup><https://github.com/junit-team/junit>. Analyzed December 2000 to September 2014.

<sup>3</sup><https://github.com/angular/angular.js>. Analyzed December 2013 to September 2014.

<sup>4</sup><https://github.com/spring-projects/spring-framework>. Analyzed July 2008 to September 2014.

<sup>5</sup>[svn://svn.valgrind.org/valgrind/trunk](https://svn.valgrind.org/valgrind/trunk) Analyzed March 2002 to September 2014.

<sup>6</sup><https://github.com/django/django>. Analyzed July 2005 to September 2014.

<sup>7</sup><https://github.com/moodle/moodle>. Analyzed November 2001 to September 2014.

<sup>8</sup><https://github.com/DragonFlyBSD/DPorts>. Analyzed October 2012 to September 2014.

The initial cloud creation times are given for the full tag cloud for the repository, which contains  $|O| + |A|$  tags. The table thus gives an indication of the cloud computation in the worst case; in practice, we can limit the number of tags shown to substantially improve this. However, the initial tag cloud is cached and so can be generated off-line in a pre-processing step. Subsequent loads of the initial tag cloud from cache are instantaneous.

Tag clouds become smaller with subsequent navigation steps and are therefore created substantially faster. Overall, navigation is instantaneous for small and medium repositories, with some degradation on the initial clouds for very large repositories with over half a million revisions.

Note that drawing the initial cloud requires us to compute the defining concepts of all objects; however, since we use an incremental lattice construction approach and therefore never compute the full lattice, we do not experience the large runtimes commonly associated with FCA.

To reduce drawing time for larger repositories we could limit the number of tags shown in the initial tag cloud to only those that apply to a larger portion of the revisions in the repository and then show the full tag set when the user has made selections to refine the tag cloud. For large repositories that are indexed repeatedly, our implementation allows us to incrementally update the context table (and therefore the concept lattice) so that updates can be performed quickly and the initial indexing need only be performed once.

## 5.7 User Study

We performed a user study in order to evaluate whether untrained users are able to answer questions about the history of a software project using ConceptCloud more or less effectively than current widely-used interfaces, the default list-view of commits or the GitHub interface, which is graph-based. Both linear list commit views, such as GitK and GitHub are widely used in practice and we therefore use these interfaces as the controls for comparison against ConceptCloud. Linear list commit views are implemented in many popular Git GUIs (such as SourceTree<sup>1</sup> and TortoiseGit<sup>2</sup>), but we make use of GitK as it is packaged standard with Git. GitK provides a searchable linear list of commits and shows the diffs between two revisions. GitHub's interface is widely used in order to visualize the history of a software project and provides graph views of a user's activity in repositories. GitHub also provides a CodeSearch interface which allows users to search for methods directly in the source code. GitK, GitHub and ConceptCloud present the same underlying information

---

<sup>1</sup><https://www.sourcetreeapp.com/>

<sup>2</sup><https://tortoisegit.org/>

through different interfaces. We therefore compare the effectiveness of our tag cloud interface to that of a searchable list interface and an interactive graph-based interface. Since the participants in our study had never used our ConceptCloud browser before, we also investigate whether the browser can be used successfully by untrained users.

### 5.7.1 Experimental Setup

We used a between-subjects design to conduct the user experiment, where each participant uses only one of the three tools to answer questions about the software development process in specified projects. We constructed three questions sets, based on three different software repositories that were also available on GitHub. All participants were asked to answer three question sets using a tool (GitK, GitHub or ConceptCloud) which was randomly assigned to them. We then evaluated the correctness of the answers supplied by the participants. Each participant was supplied with a user manual, detailing how their tool showed the history of software projects. We marked all of the answers that were submitted by the participants and calculated their results. We investigate the hypothesis that there is no difference between the correctness results obtained by the participants over all three tools.

Our user study took place in a computer lab at Stellenbosch University. All participants took part at the same time to avoid communication about the tasks. Participants were not permitted to communicate during the study.

### 5.7.2 Population

We performed our user study with students in our third year Software Engineering class of 2015. Previous courses required the students to submit assignments using Git repositories, so all were familiar with Git. The participating group consisted of 47 students in total. Participation was voluntary for all students.

No formal ethical clearance was sought for this experiment because this was a minimal risk study but ethical considerations were taken into account when designing the study. No personal or preference data was collected or stored for the participants and the question answers were stored on a password protected computer. Verbal consent was obtained from the participants after the study set up and goals were explained to them. Collected data will be destroyed at the end of the research project.

### 5.7.3 Tasks

We developed three question sets using three different repositories available on GitHub, namely RubyGems [28], Backbone [1] and Retrofit [27] (see Table 5.2).

We selected these repositories as they are popular projects available on GitHub, and they differ in size. At the time of the user study the RubyGems repository was the largest with 6388 revisions, Backbone consisted of 3130 revisions and Retrofit had 998 revisions. We used repositories of different sizes so that the results of our study would not be biased towards one repository size.

The question sets were developed by examining the repositories equally using GitK [13], GitHub [8] and ConceptCloud. Question sets included questions about the location of files, collaboration of users, expertise of the contributors as well as the history of the projects. The question answers were then verified using all three tools to make sure that the results were consistent and that each of the questions could be answered with all three tools. All questions were weighted equally. We used all three tools to generate the question sets because the different tools have different strengths and weaknesses and using only one tool would have made the questions easier to answer for the participants assigned to a specific tool. The full list of questions is provided in Table 5.2.

The GitHub interface is largely activity-based and prominently indicates the level of contribution of participants. The linear list provided by GitK easily allows for viewing the start date of a project and information about the most recent commits is displayed prominently. The ConceptCloud browse is flexible enough to answer a variety of different questions but the revision-based context provided by ConceptCloud uses the canonical representation of the repository while the file and change-based contexts might be difficult for new users to understand. For example, Question a1) is an activity-based question of which the information is prominent on GitHub and can be accessed using ConceptCloud when the right tag cloud filters are applied. The information is more difficult to observe in GitK as it would require searching for the different contributors and making notes of how many commits they have each authored. Question a6) could easily be answered using GitK as the information is provided in a temporal order and users can easily scroll to the first commit, however in GitHub this type of question is harder to answer as the GitHub commit list is paginated and users need to page through all commits to find the date of the first commit. In ConceptCloud this information can be accessed if users are able to provide the correct filters to the browser (show only date information).

Participants were given 15 minutes to answer each question set, (6, 7 and 5 questions respectively) after which they were given the next question set and corresponding repository. Participants were asked to answer as many questions as they could in the time provided and to move on from a question when they were unable to answer it.

Table 5.2: Question Set for User Study, a) Ruby Gems b) Backbone c) Retrofit

a)	<i>RubyGems:</i>
1	Who is the contributor with the most commits on the Ruby Gems project?
2	In which year were the most commits made to the project?
3	Which file types has Charlie Somerville changed in his commits?
4	Which contributors have worked on the file “lib/rubygems/psych additions.rb”?
5	Who has been making the most changes on the project since “Samuel E. Giddins” last worked on it?
6	When was this repository created?
b)	<i>Backbone:</i>
1	In which month was the most activity on the project?
2	Who was the most active developer in this month?
3	Who is the most prolific author of the “backbone/test” directory?
4	Who was the last person to change the file “backbone.js”?
5	Which file has been changed the most in this project?
6	Who has made the most changes to the images in the project (jpg, png)?
7	Who has changed the most files that “Brad Dunbar” has also changed?
c)	<i>Retrofit:</i>
1	Where are the tests for the main project located?
2	Who has edited the .yml files?
3	Who contributed the most to this project in its first year?
4	Who has worked on “JacksonConverter.java”?
5	Who merged pull request #1017?

#### 5.7.4 Analysis and Results

We used the R package for analysis of the experimental results. We performed the Shapiro-Wilk [131] test to determine whether participants’ scores were normally distributed, in order to determine what further analysis could be performed. We obtained a p-value of 0.06, and at a confidence level of 0.05 we cannot reject the null hypothesis that the data is normally distributed. We therefore conclude that the data is normally distributed.

##### 5.7.4.1 Summary Statistics

Figure 5.17 shows a summary of average correctness percentages achieved by participants for each question set, in the order that the question sets were answered (Ruby Gems, Backbone and Retrofit). Table 5.3 provides an overview of the mean, standard deviation as well as the min and max values for the average percentages that were obtained using each tool across all of the question sets.

In the first question set users of GitHub performed the best, and for the second

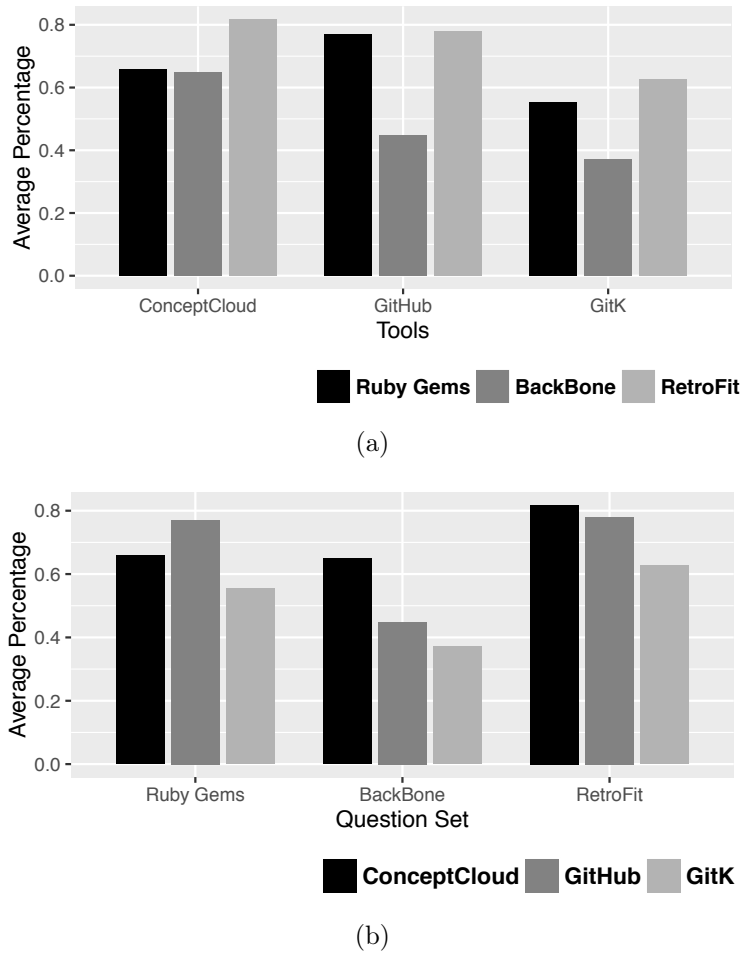


Figure 5.17: Average percentage obtained by participants, across all three tasks, using ConceptCloud, GitK or GitHub. (a) Bars from left to right indicate: ConceptCloud, GitHub and GitK (b) Bars from left to right indicate: Ruby Gems, Backbone and Retrofit Questions.

Table 5.3: Descriptive statistics for average percentages obtained with each of the three tools across all questions.

	GitK	ConceptCloud	GitHub
Mean	0.52	0.71	0.67
sd	0.21	0.10	0.1
min	0.27	0.55	0.53
max	0.84	0.90	0.85
range	0.57	0.36	0.32

and third question sets users of ConceptCloud performed the best. This could indicate that the ConceptCloud users required more time to familiarize themselves with the tool. Figure 5.18 shows a box-and-whisker plot for the average scores obtained



across all questions for each tool. We see that the median as well as the minimum value for participants using ConceptCloud is the highest, followed by GitHub and then GitK. The range of results of participants using GitK is the highest, with some participants achieving high averages and others achieving much lower results than those using either GitHub or ConceptCloud.

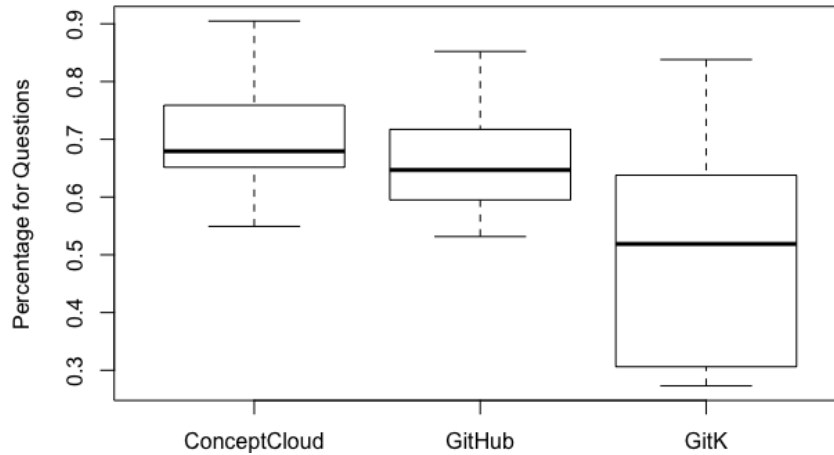


Figure 5.18: Box and whisker plots for average percentages obtained using ConceptCloud, GitK or GitHub.

Figure 5.19 shows the box-and-whisker diagrams for the percentages obtained across each of the question sets for each of the tools. Participants using ConceptCloud achieved higher median percentages for each new question set, which indicates there might have been some learning effect observed over the different question sets. However, participants using GitK or GitHub performed worse in the second question set and then again better in the third question set.

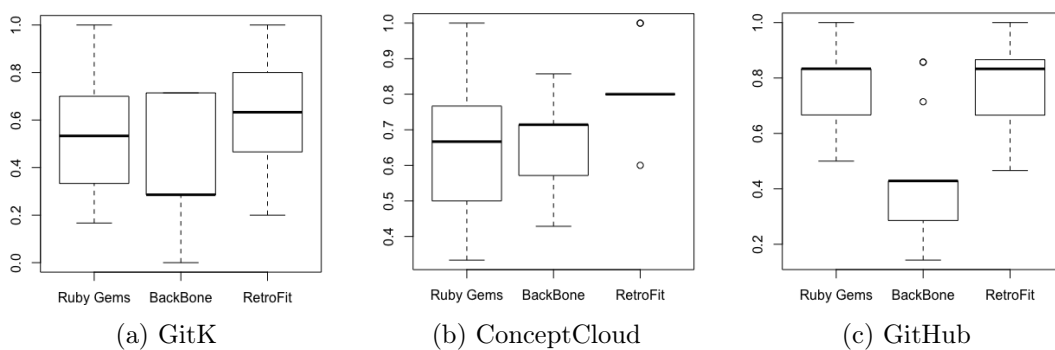


Figure 5.19: Box and whisker plots of percentages obtained by participants using (a) GitK, (b) ConceptCloud, (c) GitHub across all three question sets.

### 5.7.4.2 Statistical Significance

We performed a two-way ANOVA test on the correctness obtained by each participant across all three question sets to determine if there was any statistically significant difference in the correctness obtained by users of the different tools. We tested our null hypothesis, that the results of the participants would be the same over all three tools. We formulated this null hypothesis so that we would be able to conclude whether there was any difference in the performance of the tools, rather than only investigating whether one tool was better than another. Since we have more than two tools to compare we perform a two-way ANOVA test as opposed to a t-test as the t-test only accounts for the comparison of two tools. We first checked for interaction effects of the tools and the question sets. We found that the interaction effects were not statistically significant ( $p=0.11$ ). Therefore there is no evidence that the variation of correctness between the three tools depends on the question set. We obtained a p-value of 0.000548 from the ANOVA test for the comparison of the tools. We therefore rejected the null hypothesis at a significance level of 0.05 and concluded that the mean values of percentages obtained by participants differed statistically significantly over the three tools. We further performed a post-hoc Tukey test [142] to determine in which tool comparisons statistically significant differences exist (GitHub vs GitK etc.). The p-values obtained for all comparisons are listed in Table 5.4.

Table 5.4: p-values for Tukey test

Tool Comparison	p-value
GitHub-ConceptCloud	0.6343916
GitK-ConceptCloud	0.0006546
GitK-GitHub	0.0059474

Using a significance level of 0.05 we find that the difference between results obtained using GitK and ConceptCloud as well as GitK and GitHub are statistically significant. A graph plot of the confidence intervals is given in Figure 5.20. Therefore participants using ConceptCloud or GitHub were able to answer questions about software projects statistically significantly better than those using GitK. There was no statistical significance observed between the correctness percentages obtained by users of GitHub and ConceptCloud and it is therefore still inconclusive as to whether ConceptCloud can perform better than a graph-based interface such as GitHub's.

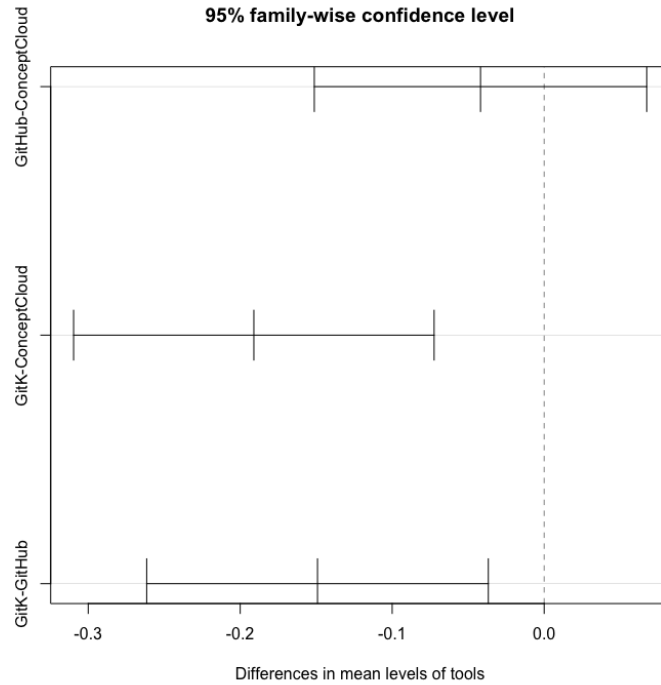


Figure 5.20: Confidence intervals obtained from Tukey Test. The GitHub-ConceptCloud interval includes 0, indicating no difference between the means for those two groups.

### 5.7.5 Question Types

We further investigated which user group had the highest result on each question to understand what types of questions were better answered through each interface.

We found that participants using GitHub were best able to answer questions about the activity on the project (“Who is the contributor with the most commits?”) as well as which users were the last to change a specific file or who has worked on a particular file. These results are to be expected as the GitHub activity charts prominently show the years and months in which the most commits have been made as well as including an activity chart for each developer. On the GitHub code search interface, specific files can be searched for and these include a list of contributors, so GitHub also makes this information prominent.

Participants using GitK were best able to answer questions about changes occurring after a developer has made their last commit as well as when a project originated. Since the linear list view provides a chronologically ordered list of commits this type of information is easy to obtain from scrolling through the commit list.

Participants using ConceptCloud were best able to answer questions about a user’s activity in a specific time period, as well as which files have been changed the most and which developers are changing certain types of files (“Who has made the

most changes to the images in the project (jpg, png)?”). ConceptCloud allows users to select a specific time period (month, year) and observe the size of the developers’ tags (commits) in this time period. ConceptCloud includes the changed file types as tags, so information about what type of work a developer is doing on a project (front-end etc.) is simple to obtain.

Therefore, while all participants were able to answer various types of questions through the different interfaces, each interface makes specific activities more prominent.

### 5.7.6 Discussion

While participants using ConceptCloud achieved the highest average over all question sets, these were only statistically significantly better than the results obtained using the linear list view provided by GitK. We can therefore conclude that the ConceptCloud interface allows users to answer questions better than a linear list view which is common in many repository GUI tools. However, no statistical significance was observed in comparison with GitHub, so we cannot conclude that the ConceptCloud interface allows users to answer questions about a software project better than a graph-based interface.

### 5.7.7 Threats to Validity

The question sets that we constructed could have been biased towards one type of visualization. However, to mitigate this risk we constructed questions using observations from all three tools equally and also verified that all questions could be answered correctly using all tools.

Our user study was conducted using a centralized lab server and so it is possible that some participants experienced slower loading times than others. However, since all of the participants took part in the user study at the same time the load on the servers would have been largely consistent for all the participants in the lab at the time.

Questions were marked by the author. However the sample solutions were verified using all tools prior to the marking process and so the questions have all been marked using answers that were consistent across all the tools.

The participants might not be representative of a real-world sample of software developers. However, all participants were also involved in their own software development projects and were familiar with Git.

Our sample size is limited, due to the size of our Software Engineering class, however, we have made conclusions from our study as much as our sample size has allowed.

The participants might have shown a bias towards our tool as it was developed at their university. However, we have never measured the participants tool preferences, but only their performance and since each participant has used only one tool (due to a between-subjects design) their performance should not be affected by any tool preferences.

## 5.8 Related Work

### 5.8.1 Visualizing Software and Bug Repositories

Zaidman et al. [155] developed a change-history view and a growth-history view to study the co-evolution of production and test code. The change history view is a plot of the changed files over the revisions of a project's repository distinguishing between production and test code. In our tag clouds we can distinguish between production and test code by observing the project's directory structure.

Girba et al. use an "Ownership Map" visualization [80] in order to identify developer interaction and development patterns using the CVS log of a project. Girba et al. also identify several behavioral patterns of developers, such as teamwork, takeover and cleaning and show how these can be identified in their ownership map visualization. These collaboration patterns could also be observed in our tag clouds constructed from a file-based context.

Alonso et al. [38] also use a tag cloud visualization to display information from version control (CVS) repositories. Their "expertise cloud visualization" creates a tag cloud of committers that are identified using rule-based classification on CVS log information. Users are then able to select the names in this cloud to display a cloud of the developer's expertise. The expertise cloud visualization [38] differs from that of ConceptCloud as the different types of information can only be displayed in separate clouds, meaning that the combinations of tags a user can select are limited, as opposed to our underlying concept lattice which only limits the available tag selections to tags that will not cause an empty tag cloud to be displayed.

Codebook [41] is a social network inspired toolset to analyze information implicitly contained in software repositories. Its central data structure is a graph, where the nodes represent the artifacts and actors (e.g., change set, developer), and the edges represent the different relations between these (e.g., contains, committer). This graph is built from different sources including revision archives, bulletin boards, mails, and directory information. Results are displayed in a web interface that provides a simple list including images of people associated with artifacts. Our context tables can also be seen as a central data structure for storing multiple types of project information.

Hipikat [57] also monitors multiple information sources (Bugzilla, CVS, email, newsgroups) and builds a uniform artifact database. It has a number of heuristics (based on text similarity and activity times) to create links between the artifacts, and provides lists of related artifacts on request. Hipikat queries are made using the Eclipse IDE and results are displayed in a Hipikat list view Eclipse plugin. However, the goal of Hipikat is more to recommend relevant items to project newcomers and not to provide them with an interface through which to explore the artifacts. Cubranic et al. [57] also note that project artifacts are not easily accessible to developers as searching the archives requires them to know the correct search terms for finding relevant information. In our work we also argue that searching the software development archives does not support all use cases, as to be able to conduct a search the developer already needs to have some information about the archive. In our approach we aim to make the information contained in software development archives accessible to users for interactive exploration so that they can access the information even before they have formulated a direct query. This is a different approach to the recommendations provided in [57] and supports users in exploring the full archives in an unbiased way.

Cubranic et al. [57] also noted that while a list-based presentation of results (as used by Hipikat) is common “when the user’s purpose is exploratory browsing of a collection, such a flat-list presentation does not indicate relationships within the results, only to the query itself.” We propose interactive tag clouds as an alternative view, as they allow users to explore query results in an aggregated form and support users in further filtering the results and identifying relationships between them.

Information Fragments [75] provide answers to developer’s questions by combining subsets of relevant project information. Information Fragments are comprised of nodes of different types, such as a team member or work item. Node types are similar to tag categories in ConceptCloud. The presentation of results in [75] uses an Eclipse plugin and supports a counting feature to get an overview of the number of occurrences of nodes, to get for example the number of items a developer has been working on. Our tag cloud automatically gives the user an overview of the number of occurrences of each item as the tags are sized according to occurrence frequency.

### 5.8.2 Tag Cloud Visualizations of Software

Guido [56] includes a tag cloud to visualize names of types, variables, parameters and methods in source code. Selecting nodes in the graph visualization that Guido also provides will highlight the corresponding tags in the tag cloud and selecting a name in the tag cloud will highlight corresponding source code elements in the graph view. The visualizations are linked in Guido similarly to the multiple tag

clouds that update simultaneously in ConceptCloud. Anslow et al. use a tag cloud to visualize the structure of Java class names in [39]. Emerson et al. use tag clouds to visualize Java methods and explore several different tag cloud layouts using the TAGGLE tool [70]. TAGGLE extends basic tag cloud views and allows highlighters to be associated with tags so that if a tag is selected related tags in the cloud will be highlighted. Tag clouds in TAGGLE are customizable, as they are in ConceptCloud, with TAGGLE additionally allowing tag layouts to be changed.

### 5.8.3 Software Engineering and Formal Concept Analysis

There have been applications of formal concept analysis to the Software Engineering domain. Tilley et al. provide an overview of these papers in [141]. Formal Concept analysis has been applied to software maintenance tasks (e.g., [99]), to identification and maintenance of structures in database schemes (e.g., [129]), to software product lines (e.g., [45]) as well as requirements analysis (e.g., [123]). We discuss here a small subset of the applications of FCA to software engineering that are comparable to our approach.

Poshyvanyk and Marcus [117] use a combination of latent semantic indexing and concept lattices to find methods that are relevant to a bug report. Girba et al. [79] use concept analysis to detect co-change patterns in revision control systems. Objects are packages, classes, or methods, while properties are the validity of expressions over certain metrics of the objects (e.g., number of classes, methods, or statements); the specific expression is determined by which co-change pattern is to be detected. Similar ideas could be integrated into our approach.

There have also been direct applications of formal concept analysis to source code analysis and re-engineering [136, 137] but these only consider an individual program, not a repository.

## 5.9 Conclusion

In this chapter we have applied our interactive browser to data contained in revision control archives. Our browser can then be used to answer many difficult questions such as “What has happened in this project while I was away?”, “Which developers collaborate?”, or “What are the co-changed methods?”.

By changing the type of object in the context table (e.g., revision, file etc.) we are able to provide complementary views on the same underlying data and observe collaboration patterns of the developers. By using changes (i.e., revision-file pairs) as objects we are able to easily identify the co-changed methods in a project. Addi-

tionally, our context tables can be used as a centralized data structure for multiple sources of information, such as version control data and bug reports.

Our tag clouds provide a visualization in which version control data can be aggregated and explored interactively to support developers in tasks such as keeping up with project changes. Our interactive visualization supports users in exploratory search tasks when they have no previous knowledge of a project.

We have used the ConceptCloud browser to repeat a previous case study [148] and to make observations about the internal structure of a small commercial development project. We have also performed a user study to determine the usability of ConceptCloud and to compare its effectiveness in allowing users to answer historical questions about a project to that of other existing information representations. Through our user study we conclude that new users are able to make use of our ConceptCloud browser to answer questions about the history of a software project.



## Chapter 6

# Concept-Based Exploration of Developer Skill Sets Identified from Their Open Source Contributions

In this chapter we describe how we have used the ConceptCloud framework to build the CVExplorer tool, which allows users to browse skills information extracted from a collection of projects on GitHub. CVExplorer can be used to filter a large pool of developers by selecting relevant skills that the developers are required to have. We initially conducted a survey of GitHub users and recruiters in order to evaluate what role GitHub data currently plays in the hiring process. Through our survey we learned that recruiters experience barriers to entry when using GitHub data in the recruitment process and therefore, developers are often contacted about positions for which they are not qualified. Developers mentioned that recruiters often make mistakes when assessing their GitHub profiles. CVExplorer has been designed to make skills information for developers on GitHub more accessible so that assessment of a user's skill set can be done more accurately, leading developers to receive less contact about unsuitable job openings. We describe our CVExplorer tool in this chapter and use it to recommend candidates for open positions at two companies and ask for feedback as to whether the candidates were suitable to interview.

### 6.1 Introduction

Poor hiring decisions are a well-known risk factor in the success of a software project [138]. DeMarco and Lister observe that work quality is more dependent on the team

members involved than on how the work is done [63]. In industries such as Software Engineering with many open positions and not as many qualified candidates, identifying (or *sourcing*) candidates with the right combination of skills is crucial to the success of a software project because these candidates may not actively be applying for jobs themselves [106]. Developers' open source contributions have been suggested as a mechanism to determine suitability for a particular job [94, 104]. Open source contributions allow us to infer information about the developers' interests (e.g., what they program in their free time) and technical skill sets (e.g., which programming languages they are using) which can both be used to improve the quality of candidate sourcing. There are even claims that "GitHub is a developer's new CV" [65, 147].

In order to evaluate the role that GitHub currently plays in hiring software developers, we surveyed 85 developers on GitHub on their experience of the use of GitHub in the hiring process. We asked developers how they perceive contact from recruiters and whether their GitHub profiles played a role in their current or previous employment. Developers note that their open source contributions are not fully utilized to personalize and reduce unnecessary contact from recruiters, which would benefit both the developer and the recruiter. Developers also experience that recruiters are prone to misinterpreting profile information. We surveyed seven recruiters and informally interviewed six company recruitment representatives to ask *specifically* how the recruiters use GitHub for candidate identification. Our recruiter survey responses corroborate those of Singer et al. [133] where recruiters mentioned struggling to interpret information on code sharing sites.

We then constructed the CVExplorer tool (which is available at <http://recruit.conceptcloud.org>) using our ConceptCloud browser framework to present technical skills extracted from GitHub data in an intuitive, interactive tag cloud interface. Our tag clouds make GitHub data more accessible for skills-evaluation tasks. Our approach is novel in that it allows users to explore skill sets of a large number of developers *simultaneously* and narrow the developer pool to only those that possess relevant skills. However, our interface can also function as a skill aggregator: once a candidate has been identified, the individual candidate's full set of skills are displayed, which can be used to personalize contact with the candidate.

Business-oriented social networks such as LinkedIn [19], where profiles are self-authored, are commonly used for online recruitment. Individuals may exaggerate their skill-set or, conversely, omit some skills in a self-authored CV. Therefore, diversifying the candidate search to developer-oriented sites such as GitHub, is beneficial to the recruitment process. Additionally, as noted by Capiluppi et al. [44], a skills-based identification could provide equal opportunities to skilled developers that have no formal qualifications.

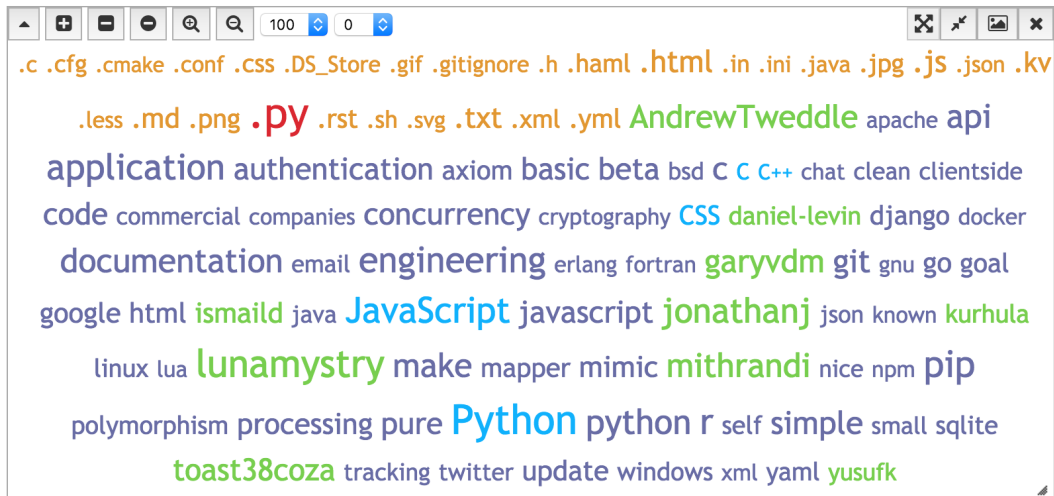


Figure 6.1: Skills (purple), filetypes (orange), and developers (green) associated with changes to Python files (selected, appears in red) for Cape Town developers on GitHub.

There are already sites (such as Coderwall [4], Masterbranch [24], Open Hub [25], Stack Overflow Careers [30] or TalentBin [32]) which aggregate developers' skills across a site or on various platforms, but these typically assist in evaluating individual candidates that have already been sourced through other means or have actively applied for a position. Therefore, CVExplorer serves a *different purpose* to profile aggregation which only allows evaluation of the skills of one developer at a time and does not allow multiple developers to be filtered to those that have suitable skill sets.

CVExplorer supports *identification of suitable technical candidates* from a large pool of developers as opposed to aggregation of individual developers' skills across platforms. We mine and aggregate *technical skills* across all of a developer's commits and present these in an interactive tag cloud (see Figure 6.1) in order to facilitate skills browsing and identification of suitable developers. Users can filter the developer pool by selecting a combination of tags that comprise relevant skills.

GitHub contributions are already compared to a traditional CV as serving a summary of the developer's skills and therefore, we stick to the terminology of a "CV". Note that this CV is not in the traditional form of activities in chronological order, but is similar to a portfolio of the developers' skills. However, code contributions constitute the actual portfolio and so we extract the portfolio's meta-information.

In this chapter we investigate the following research questions:

- RQ1: What role do GitHub contributions play in the hiring process, specifically, do recruiters note GitHub contributions when they contact technical candidates?
- RQ2: Can GitHub *technical skill* data (e.g., programming languages and databases)

be *meaningfully* mined and aggregated to build an open source CV for a GitHub user?

RQ3: Can CVExplorer be used to *successfully* identify candidates suitable for open positions?

In order to determine the effectiveness of CVExplorer we used it to recommend candidates for positions at two companies; we received positive feedback on the candidates' suitability for interviews.

## 6.2 Background

### 6.2.1 Candidate Identification

Job candidates are referred to as either “active” or “passive” [106]. Active candidates are actively applying for jobs and passive candidates are being considered for jobs that they have not applied for [62, 106]. In order to hire passive candidates a recruiter needs to find and evaluate candidates, before contacting them about available positions. In industries such as Software Engineering where there are many open positions and not as many qualified candidates, sourcing passive candidates becomes important because high-quality employees may not be applying for jobs themselves [106].

Traditionally, sourcing was done by making phone contact with candidates and was also heavily recommendation-based. Now recruiters also search the Internet for CVs and browse business-oriented social networking sites such as LinkedIn.

### 6.2.2 Interpreting User Profiles

Marlow and Dabbish [104] discuss *signals* for certain characteristics as cues which are interpreted from a user's profile. They investigate which activities serve as signals for developers on open-source hosting sites. Signals gathered from a user's profile may be assessment signals or conventional signals. Assessment signals are harder to fake while conventional signals can easily be faked [104] and are therefore not reliable and need to be verified. For example, an assessment signal could be lots of check-ins of Java code to a project on GitHub which indicates that a developer has Java experience. A conventional signal could be simply forking a Java repository which is easy to do to fake Java experience.

We use the term signals to refer to aspects of a developer's profile used to assess the developer that may not be carefully verified.

### 6.2.3 Business-Oriented Social Networks

Business-oriented social networks such as LinkedIn and Xing started in 2002. We refer to LinkedIn as it is by far the largest social network of this kind [20]. Users enter online CVs on these sites and can connect with other users. Other site users (not necessarily directly connected) can then view a user's profile and, depending on the account tier, send the user a message which is displayed on LinkedIn. Companies can post job listings on LinkedIn and search the user base. LinkedIn also runs a section of the website which is dedicated to recruiters and providing "talent solutions" which include additional search filters, such as years of experience and current position. In addition, LinkedIn Recruiter also provides suggestions of potential candidates based on the recruiter's previous searches [21]. LinkedIn profiles (for example, Figure 6.2) also list skills that the user has added to their profile and that can be endorsed by other users; these profiles are therefore also based on the user's reputation.

### 6.2.4 Open Source Hosting Sites

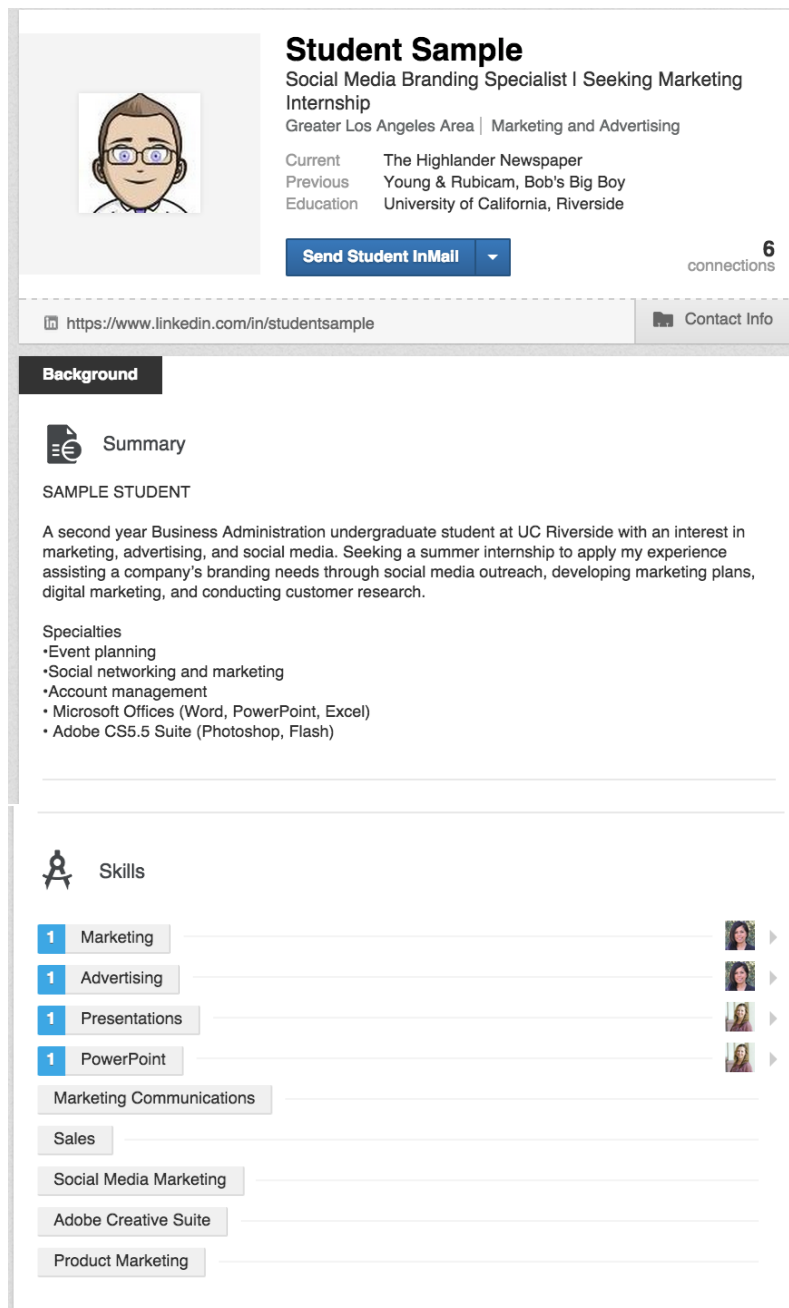
Open-source hosting sites began in 1999 with SourceForge. With the increased popularity of the distributed version control system Git [100], hosting sites GitHub [8] and Bitbucket [2] have become the standard for open source projects.

We use GitHub in our study as private GitHub repositories are limited to paying customers and therefore GitHub has become synonymous with open source development. It is also the most popular open source hosting site with more than 10.9 million users [12]. GitHub allows users to host their Git and Mercurial software repositories, "fork" other users' repositories to copy them to their own profiles and create pull requests to incorporate their code into other users' repositories. In addition to this functionality, GitHub allows users to provide comments on specific lines of code and pull requests, contains an issue system and provides statistics on users' contributions to projects.

### 6.2.5 Profile and Skill Aggregators

Developers often have multiple profiles on different sites containing different information about their interests; a more complete picture of a developer can be obtained by combining this information. Various profile aggregators merge profiles on various social platforms in order to facilitate social recruiting. Some commercial profile aggregators include Coderwall, Masterbranch, Open Hub and Talentbin.

Coderwall [4] is a profile aggregator for Twitter, Facebook and GitHub, while Masterbranch [24] aggregates developers' repositories and shows what percentage of the code in the repositories is in a particular language. Open Hub [25] provides



The image shows a LinkedIn profile for a user named 'Student Sample'. The profile includes a profile picture of a man with glasses, a title 'Social Media Branding Specialist | Seeking Marketing Internship', and location 'Greater Los Angeles Area | Marketing and Advertising'. It lists current employment at 'The Highlander Newspaper', previous roles at 'Young & Rubicam, Bob's Big Boy', and education at 'University of California, Riverside'. There is a 'Send Student InMail' button and '6 connections' listed. Below the header is a 'Background' section with a 'Summary' icon. The summary text reads: 'SAMPLE STUDENT. A second year Business Administration undergraduate student at UC Riverside with an interest in marketing, advertising, and social media. Seeking a summer internship to apply my experience assisting a company's branding needs through social media outreach, developing marketing plans, digital marketing, and conducting customer research.' It lists specialties: 'Event planning', 'Social networking and marketing', 'Account management', 'Microsoft Offices (Word, PowerPoint, Excel)', and 'Adobe CS5.5 Suite (Photoshop, Flash)'. The 'Skills' section shows a list of skills with progress bars and endorsement icons: Marketing, Advertising, Presentations, PowerPoint, Marketing Communications, Sales, Social Media Marketing, Adobe Creative Suite, and Product Marketing.

**Student Sample**  
Social Media Branding Specialist | Seeking Marketing Internship  
Greater Los Angeles Area | Marketing and Advertising

Current The Highlander Newspaper  
Previous Young & Rubicam, Bob's Big Boy  
Education University of California, Riverside

Send Student InMail

6 connections

<https://www.linkedin.com/in/studentsample> Contact Info

**Background**

**Summary**

SAMPLE STUDENT

A second year Business Administration undergraduate student at UC Riverside with an interest in marketing, advertising, and social media. Seeking a summer internship to apply my experience assisting a company's branding needs through social media outreach, developing marketing plans, digital marketing, and conducting customer research.

Specialties

- Event planning
- Social networking and marketing
- Account management
- Microsoft Offices (Word, PowerPoint, Excel)
- Adobe CS5.5 Suite (Photoshop, Flash)

**Skills**

- 1 Marketing
- 1 Advertising
- 1 Presentations
- 1 PowerPoint
- Marketing Communications
- Sales
- Social Media Marketing
- Adobe Creative Suite
- Product Marketing

Figure 6.2: Example of a LinkedIn profile from [22] which shows the user's profile picture and a summary of their current position as well as a list of skills which can be endorsed by other LinkedIn users.

a profile for open-source developers, which indicates the number of commits by a developer in a time period, as well as presenting a summary of the programming languages.

Talentbin allows users to search developers with skills of interest in a particular location, therefore unlike previous profile aggregators TalentBin is optimized for candidate identification. Talentbin [32] describes itself as a “talent search engine” aimed at finding technical candidates. Data on Talentbin is sourced from profiles on Twitter, Facebook, GitHub, LinkedIn, Quora, Meetup.com, the US patent database, and other platforms. Talentbin reports what repositories users own on GitHub and what actions they have taken (such as starred a project); however, it is unclear whether Talentbin analyses individual commits of users to repositories. The developer interests extracted by Talentbin are also signal-based [104]; for example, one user showed up with an interest of “Reading” because they had a project with a description containing the text “Reading MP3 files”.

Stack Overflow also provides a “Candidate Search” [29] which is a paid service that allows recruiters to search for candidates with particular experience (derived from the user’s profile tags). Stack Overflow’s candidate search is able to facilitate developer sourcing but the information is limited to that available within the site and provided by the user.

## 6.3 Role of Open Source Contributions in Hiring Technical Candidates

In order to determine the role of open source contributions on GitHub in the hiring process we surveyed both developers and recruiters. We asked recruiters if and how they use GitHub in order to source technical candidates and we asked developers how often they were contacted by recruiters via GitHub and what their experience of this contact was. Our recruiter survey has only seven participants with a response rate of 4.8% while our developer survey had 85 respondents with a response rate of 7.5%.

### 6.3.1 Study Design

#### 6.3.1.1 GitHub Users

We obtained a random sample of GitHub users from the GitHub Archive [11] which contains GitHub event data and makes this queryable via Google Big Query [15]. We limited our sample to developers that provided an email address on their profile so that they would be contactable. We contacted developers in batches over multiple

Table 6.1: Developer survey questions posed to developers on GitHub. \* indicates a required question. Where question answers were provided as free text this is also indicated at the end of the question.

	Question
1	Have you received an email from a recruiter explicitly stating that they found you on GitHub? e.g "Hi, I found your GitHub profile..." *
2	Has your GitHub profile played a role in you getting a job in the past? Please give details. (free text answer)
3	Do you also have a LinkedIn account *
4	How often are you contacted by a recruiter on any social site? * (free text answer)
5	For what reason to you provide your contact information on your GitHub profile? (free text answer)
6	Do you have any other comments about contact from recruiters or your use of GitHub? (free text answer)
7	In which city are you? (Optional) (free text answer)

days (due to survey sending restrictions on Google Forms) and asked them to fill in a short survey about recruitment of GitHub users. In total we contacted 1140 developers by email and received 85 responses, a 7.5% response rate.

We asked developers how often they are contacted by recruiters where the email specifically references the user's GitHub profile, whether GitHub had played a role in their previous or current employment and whether they were also members of LinkedIn. We further asked developers to provide comment on their sentiments towards contact from recruiters. We received responses from developers in 25 different countries, with the largest group of 25 developers coming from the USA. Full survey questions are available in Table 6.1.

No formal ethical clearance was sought for this survey because this was a minimal risk study but ethical considerations were taken into account when designing the study. We used Google forms to conduct the survey to ensure that participation in the survey was anonymous. Participation in the survey was strictly voluntary. No identifiable personal information was collected from any of the participants. Collected data will be destroyed at the end of the research project.

### 6.3.1.2 Recruiters

In order to complement the results from the developer survey we manually identified 144 international recruitment companies through web searches for technical recruitment companies and emailed them to ask them to fill out a survey with questions on their use of GitHub. The full question list of the survey can be found in Table 6.2. We received seven responses (4.8% response rate). Our sample size is comparable to



that of Singer et al. [133] who interviewed 13 recruiters in order to determine how they use social sites and profile aggregators.

At the Stellenbosch University Computer Science career day (held on August 26th, 2015) we informally interviewed six company recruitment representatives about their use of GitHub in recruitment with the same underlying goals. We asked the respondents whether they used GitHub in the hiring process and whether they were aware of GitHub facilitating the sourcing of candidates. We also asked recruiters if they were interested in a developer's fine-grained skill set (such as recruiting Android developers, or developers with experience on particular frameworks such as Ruby on Rails) and whether they examine a developer's code samples on GitHub once they have been sourced on other platforms.

No formal ethical clearance was sought for this survey because this was a minimal risk study but ethical considerations were taken into account when designing the study. We used Google forms to conduct the written survey to ensure that participation in the survey was anonymous. Participation in the survey and informal interviews was strictly voluntary. No identifiable personal information was collected or stored for any of the participants. Collected data will be destroyed at the end of the research project.

## 6.3.2 GitHub User Survey Findings

### 6.3.2.1 Recruiter Contact

We asked respondents to indicate how many times they were on average contacted by a recruiter where the recruiter specifically mentioned the respondent's GitHub account. This question was required and so all participants answered it. 55% of respondents noted that they have been contacted via GitHub at least once, with 25% reporting that this had only been between 1 and 5 times, 15% reporting they were contacted between 5 and 20 times and only 6.7% reporting they were contacted more than 20 times. 8.3% reported receiving contact mentioning their GitHub profiles only once. Note that respondents did not differentiate between recruiters employed by a recruitment company, in-house recruiters employed by a development company or other company employees (such as developers) fulfilling a recruitment role.

### 6.3.2.2 Other Social Platforms

Of the total respondents 71 (83.5%) indicated that they had a LinkedIn profile and 14 (16.5%) did not. 18.4% of users that were not contacted via GitHub indicated that they did not have a LinkedIn profile and 14.9% of users that were contacted via GitHub indicated that they did not have a LinkedIn profile. Of users that had never

Table 6.2: Survey questions posed to recruiters about their use of GitHub in the sourcing and hiring processes.

	Question
1	Do you use GitHub at all in the recruitment process? *
2	Do you search GitHub for developers in your region?
3	If you find a developer on GitHub what attributes do you look for on their profile? (free text answer)
4	Have you successfully matched a developer found on GitHub to a job opening?
5	Do you begin your search for developers by using GitHub or do you use it only as a secondary resource? Please comment on this. (free text answer)
6	<p>What attribute do you find most important on a GitHub profile?</p> <ul style="list-style-type: none"> <li>• Amount of profile activity, such as starred and forked projects, reporting and commenting on issues.</li> <li>• Content of commits - what kind of contribution this person has been making</li> <li>• Programming languages of the developer's repositories</li> <li>• Commit activity - number of commits the person is making to any projects</li> <li>• Other:</li> </ul>
7	Do you have any particular problems when finding developers on GitHub? (free text answer)
8	Do you compare a developer's LinkedIn profile to information obtained on their GitHub profile? e.g., if Java is listed as a skill on LinkedIn do you check this against the candidate's open source contributions? If so, please explain when you would be likely do this. (free text answer)
9	Are you interested in a developer's fine-grained skill set extending beyond programming languages (such as databases, PostgreSQL, MySQL, web frameworks such as Ruby on Rails, Spring Framework etc.)? (free text answer)
10	Do you currently use GitHub to obtain a developer's fine-grained skill set (not just the programming languages)? (free text answer)
11	Do you look at the code that a developer has committed to projects? (free text answer)
12	It has been claimed that GitHub is a developer's new CV. Do you agree/disagree with this statement and why? (free text answer)
13	If a developer has forked a Java repository on GitHub but never made any contributions to the repository, would you take this an indication that the Java is one of the developer's skills? (free text answer)
14	Is there any additional support for finding developers that you would want GitHub to provide? (free text answer)
15	Please provide any further comments on your use of GitHub here (free text answer)
16	Please indicate in which city you are working * (free text answer)

been contacted on GitHub 71.0% reported being contacted by recruiters on other social sites. This indicates that recruiters were contacting these candidates but not evaluating or not making mention of their GitHub profiles.

### 6.3.2.3 Developer Sentiment on Contact From Recruiters

We asked respondents to provide comments on the contact that they receive from recruiters. The overall theme of the comments was that respondents feel that recruiters do not take enough time to evaluate their open source contributions before contacting them about job opportunities:

*I really appreciate recruiters that have taken the time to at least skim through my projects on GitHub, but most still email me without researching me at all. (R81)*

*They often do cursory evaluation and don't pay attention to what I'm really doing. (R66)*

However, information about the number of commits to projects and the skills demonstrated in contributions is either very time consuming to obtain on GitHub, or limited only to users with a strong technical background that evaluate the content of a developer's commits. One developer comments that recruiters make mistakes when interpreting his profile:

*Most of them are non-technical and make mistakes that are very noticeable, or don't look at my projects in detail enough to say meaningful things about them. (R83)*

Other developers note that recruiters have difficulty gathering sufficient detail on GitHub:

*Although the recruiter had an overview of what languages I program in in my free time, and it impressed me that they got this information from GitHub, they seemed to have only a very shallow overview and hence an inaccurate idea of my skills. (R72)*

*[M]ost of them don't read anything on GitHub, they just stop at the left section of the profile (stars/languages/contact). Recruiters simply need to read stuff before contacting people. (R23)*

One respondent indicated that they liked the fact that recruiters approached them for job openings and indicated that he

*[...] never actively looked for a job, they always offered me a position.  
(R1).*

#### 6.3.2.4 Reasons for Providing Contact Information on Profile

We asked respondents why they provided their contact information on their GitHub profiles, since this is an optional field. We received 66 answers to this question. Candidates responses typically mentioned that they provided their contact information to receive contact about their open source projects, from users or collaborators; however some (19.7%) candidates specifically provided their contact information in order to receive emails about job opportunities.

*It can help recruiters who search for candidates based on their projects/-  
work contact me in an easy way. (R61)*

*To get emails from recruiter's (Because this might be helpful in the future)  
(R4)*

*For users or collaborators of my open source projects to reach me directly  
(R66)*

#### 6.3.2.5 Role of GitHub in Respondent's Past and Current Employment

Of all 85 respondents only 66 answered this question. Of all answers 36.5% indicated that their GitHub account had played a role in them receiving employment in the past. Answers ranged from candidates indicating that they were asked for their GitHub profile during the interview process to one candidate mentioning they

*[...] got my present job by having a complex project on GitHub in the  
exact framework that the company was interested in. (R19).*

*Basically GitHub acts like my portfolio. Every time I apply for a job  
people ask for my GitHub profile. (R16)*

*Yes, I was approached for my current job based primarily on my GitHub  
profile. (R50)*

However other candidates supplied their GitHub profiles along with their CVs but were unsure what impact this had.

*I've given companies links to my projects, but I do not know if it has ever  
been visited. (R75)*

### 6.3.3 Recruiter Survey and Interview Findings

We spoke to six technical recruitment representatives from companies about their use of GitHub for recruitment purposes and received survey responses from seven international recruitment agencies involved in recruiting technical candidates. Only one of the surveyed recruiters indicated that they had been able to successfully match a developer sourced on GitHub to a position. However, surveyed recruiters did indicate that they used GitHub as a secondary resource to evaluate candidates:

*[W]e suggest to our clients that they look at the candidate's profiles on GitHub (C2).*

Another recruiter from a recruitment company indicated that they sourced candidates through a

*Personal Database, linkedin etc and then I check github of each candidate I talk to, to see how active they are (C6).*

None of the recruitment representatives from companies that we interviewed indicated that they used GitHub at all in the hiring process.

Our recruiter responses reinforced sentiments expressed in the recruiter survey of [133] that recruiters can have difficulty interpreting open source code hosting sites. Our responses indicate that recruiters who are using GitHub in the hiring process are mainly evaluating candidates that have been sourced on other platforms. Therefore, even if developers receive contact from a recruiter that notes their GitHub profile, this does not necessarily indicate that the recruiter sourced the candidate on GitHub.

Recruiters also indicated that they were interested in obtaining a developer's skills (such as databases used etc.) when they were sourcing candidates.

### 6.3.4 Survey Summary

#### 6.3.4.1 GitHub Users

From our developer survey responses we learn that: open source developers are aware that their contributions can serve as a CV and are willing to have their contributions used in this manner. Open source developers *expect* recruiters to use the information provided on their open source profiles in order to determine if they are a suitable match *before contacting them* about available positions.

#### 6.3.4.2 Recruiters

From our recruiter survey responses and interviews we learn that not all recruiters have awareness of the features provided on GitHub that would allow for candidate

identification (e.g., advanced search) and are therefore not making use of these in their day-to-day hiring activities. Recruiters that are making use of GitHub in the hiring process are mainly using it to evaluate candidates that have been sourced on other platforms.

#### 6.3.4.3 Technical Aspects

We also identify the technical challenges for identifying candidates using GitHub, which are highlighted in both our developer and recruiter surveys. In particular, there is a technology barrier to the use of GitHub in identifying candidates. Developers indicate that recruiters using GitHub are prone to misinterpreting user profiles. Some developers experience recruitment negatively and are frustrated by the recruiters lack of correct interpretation of GitHub data resulting in impersonal contact. GitHub itself does not make information about what types of changes users are making, (i.e., their skills and expertise) easily accessible and does not provide aggregations or support for identifying candidates as part of a user's profile.

#### 6.3.4.4 GitHub in the Hiring Process

Investigating RQ1, we discover that GitHub is used in the hiring process and recruiters note a candidate's GitHub profile when contacting them. However, recruiters that do use GitHub indicate that they mainly use it to evaluate candidates that have been sourced on other platforms and not as a first step in candidate identification. Therefore, developer and recruiter responses indicate that GitHub is not being fully utilized in order to correctly identify candidates or in order to evaluate a specific candidate's skills accurately.

## 6.4 Presenting Developer Contributions in a Browseable Format

In order to break down the technology barrier identified in our surveys we construct CVExplorer using GitHub profile data to make multiple developers' skill sets simultaneously browsable and present their skills extracted from their open source contributions in an aggregated form.

CVExplorer is built on top of the core of our ConceptCloud browser (see Chapter 4). Here we adapt the tool to display information from multiple repositories and to include technical skills and programming languages extracted directly from GitHub.

We process the developer's commits over multiple repositories off-line and load this into ConceptCloud to be visualized as cloning multiple repositories can be too

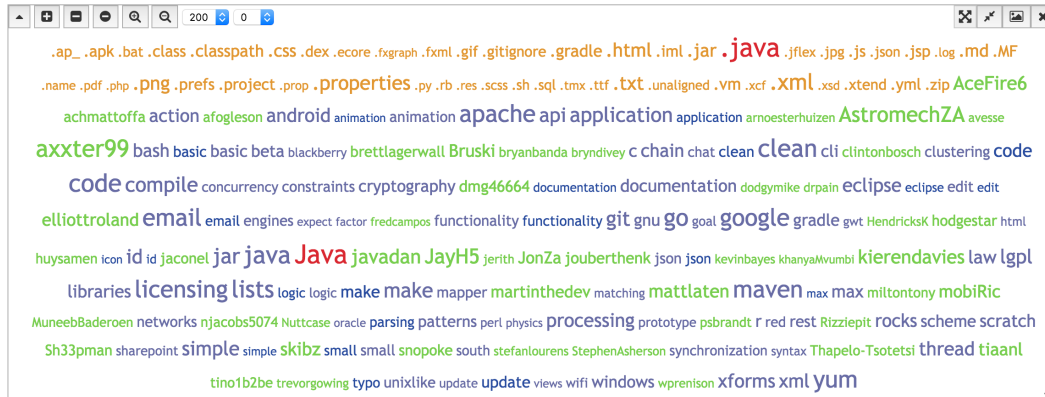


Figure 6.3: 200 largest tags from commits in which `.java` files have been edited in Java projects; developer usernames are indicated in green, selected tags are indicated in red. Large green tags indicate developers with many Java file changes (and presumably therefore good Java expertise).

time consuming online (e.g., our Cape Town tag cloud took roughly six hours to compute).

#### 6.4.1 Evaluating Skills via GitHub’s Interface

A developer’s GitHub profile contains information about the number of followers they have, their repositories and their public activity (e.g., following another developer or starring a project). GitHub provides an activity chart indicating how many commits a user has made over the previous year. This information is prevalent on a user’s profile and some survey respondents mentioned that this is the only information recruiters seem to assess on their profiles.

On a GitHub profile, information about the content of the changes a developer has been making is significantly more difficult to obtain than their activity. GitHub profiles link all projects that the user has contributed to or forked but do not directly provide any additional information about the contribution. In this format the profile can only be considered as a signal [104] for the developer’s interests or expertise. Inexperienced GitHub users (such as non-technical recruiters) might not be aware that the information on the user’s profile needs to be verified by looking at detailed information about the user’s commits to verify what contributions they are making.

There are a number of shortcomings on GitHub profiles when attempting to assess a developer’s skills from the profile. In order to get an accurate representation of a developer’s skills via the GitHub interface, a user needs to select all projects that the developer has contributed to and verify the contributions. Programming languages for all projects then need to be manually aggregated to get an overview of the developer’s skills set. However, even a developer contributing to a Ruby project

may not have changed any Ruby code in his contributions and so the developer's skills need to be manually verified on an individual commit level.

On GitHub it is also non-trivial to find developers using a particular programming language. GitHub does provide an advanced search feature that allows users to find developers that use a particular programming language, but this still does not indicate how much experience the developer has in that language and what other languages they are experienced in.

### 6.4.2 Mining Developer Contributions

We mined developer profiles according to the location provided on their GitHub profiles as recruiter respondents indicated that they tried to recruit developers from a specific location for positions available in that location. We used the GitHub API [10] (more specifically the Eclipse EGit GitHub reader [6]) in order to obtain the most up-to-date information for 1000 developers in a specific location and to extract a list of each developer's repositories. We extracted the first 1000 developers that were available from the API (which limits search results to 1000 responses). We then automatically cloned all repositories to which those 1000 developers contributed and identified those developers' individual commits. We extracted from each commit, the commit message and the changed files directly from the Git repositories using the Eclipse JGit library [5]. Since the commit logs do not always show the author's GitHub username, we associated developers to commits by automatically matching the commit name to either the developer's listed name (i.e., the name they add to their GitHub profile which may or may not be their full name) or username on GitHub. For each repository we also extracted the programming language as it was listed on GitHub (using the GitHub API) and the project's ReadMe file. ReadMe files contain details about the project such as the type of technologies it makes use of, installation instructions and the project's dependencies. Therefore, the ReadMe file can provide an overview of the project and indicate what skills a developer contributing to the project is likely to possess.

We associated a project (and its extracted information) to a developer only if the developer had made more than five commits to the project, in order to ensure that the developer was not only a one-off contributor.

### 6.4.3 Aggregating Developer Contributions

We aggregate all of a developer's commits to all of their repositories on GitHub and identify in particular the file types that the developer has changed. We also process the ReadMe files of all the developer's repositories in order to extract skills that



a contributor to the project is likely to possess (experience with a particular web framework or database).

We constructed a white list of skills (see [www.conceptcloud.org/hiring\\_from\\_github](http://www.conceptcloud.org/hiring_from_github)) from the Wikipedia [16] lists of Programming Languages, Web Frameworks, Platform Independent GUI Libraries, Ajax Frameworks, Object-Relational Mapping Frameworks as well as the ACM Classification specifications. This white list allows us to extract “skills” or technologies used as well as high level phrases such as “Machine Learning” from the ReadMe files of projects that a developer has contributed to. We run each project’s ReadMe file through the white list in order to generate a bag of words containing the extracted skills that appear on our whitelist for each project.

We also ran the commit message through our white list of skills in order to filter out any words that do not directly reference a technology or concept (such as “added” etc.). We removed punctuation and spaces when matching skills on our whitelist to text in the ReadMe files in order to ensure that we pick up all possible matches. We then aggregated the identified skills for each commit and presented the information in a browsable format.

Note that while skills identified in a ReadMe file could be termed as “conventional signals” [104] because these are easier to fake (adding additional technologies to the ReadMe), we include the changed file types for each commit because these can be considered as “assessment signals” [104] as they are harder to fake. The user can verify that a developer has been making changes to `.java` files before the developer is considered to have Java as a skill.

#### 6.4.4 Presenting Developer Contributions in a Browsable Format

We visualize developers’ contributions and skills in an *interactive* tag cloud which can be refined by selecting relevant skills. Figure 6.3 shows a tag cloud of the 200 largest tags from our sample of developers in Cape Town that have changed `.java` files. The developer tags are indicated in green. Figure 6.3 shows that some skills associated with changes to Java files are **Android**, **Maven** or **Apache**. We also see that developers changing `.java` files have also changed `html`, `gradle` and `XML` files. Selecting tag **Android** in the cloud would further restrict the cloud to showing only developers that have changed Java files in a project that mentions Android in its ReadMe file, and other skills exhibited in the changes to these projects.

Skills tags are sized according to the *number of commits* in which the developer has exhibited the skill (such as changing a `.java` file) as opposed to the *number of files* they have changed in one particular commit, e.g., if a developer changes multiple `.java` files in one commit `.java` is still only associated to the commit once. Therefore a larger sized tag indicates that a skill is exhibited consistently as opposed to being

exhibited many times in only one commit. Note that a skill exhibited in 100 commits to the same project will be sized the same as a skill tag that is exhibited in individual commits across 100 projects. However, since the project name is present as a tag in the cloud as well, it is easy to identify whether the developer has exhibited a skill in multiple projects (observing multiple project tags, cf. Figure 6.5(c) where project tags for the selected developer are indicated in pink) or only one (observing a single project tag).

Particular web frameworks are associated with a group of file types (e.g., Ruby on Rails with `.erb`, `.rb`, `.html` and `.css`). Therefore, multiple file types can be selected in the tag cloud to reveal developers that use a particular framework.

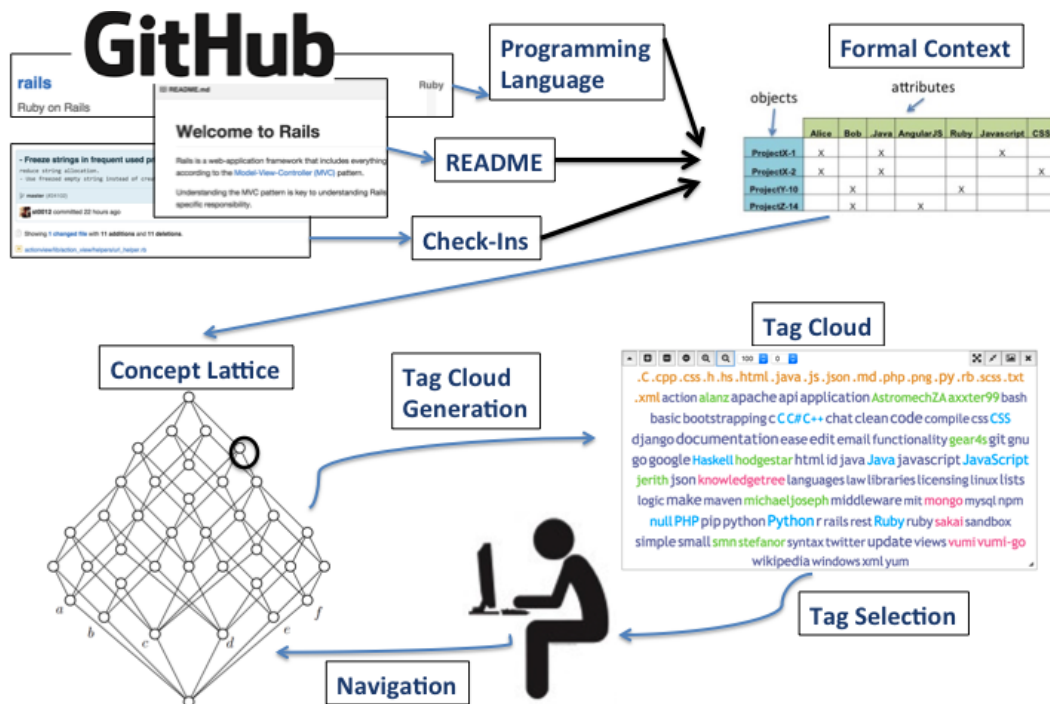


Figure 6.4: Context table and concept lattice construction from GitHub README, programming language and checkins. Navigation is driven by the user’s tag selections which update the focus in the lattice and generate a new tag cloud, denoted by the circular process.

Figure 6.4 shows the process of constructing a tag cloud from information extracted from GitHub. We extract information from the README, GitHub programming language and check-in information (changed file types and commit messages) to construct the formal context. As objects (rows) in the context table we use the project’s name followed by the commit number of the check-in, such as “projectX-1”. Therefore the objects in this context are still individual revisions such as described in Section 5.3.2, however, instead of using all revisions of a selected project as the

objects in the context, here we use revisions from a large collection of projects in the same context. We use as attributes all skills derived from the project and commit messages as well as changed files from the check-in.

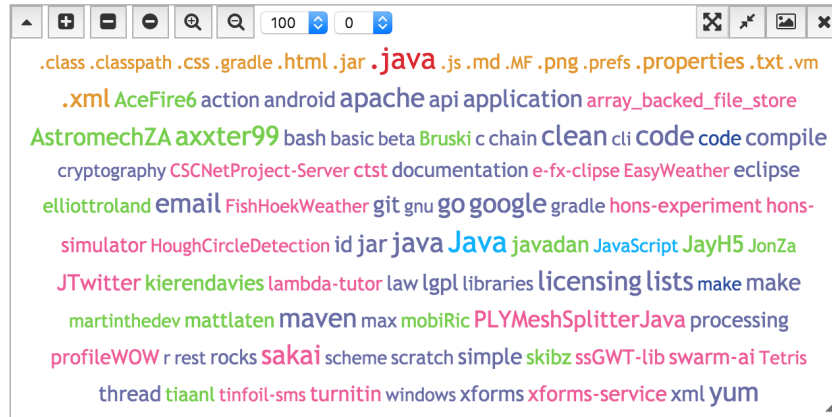
Figure 6.5(a) shows a tag cloud with the tag for a Java file type selected (`.java` indicated in red). We then add the tag for a specific developer in 6.5(b) to see how many times this developer has changed `.java` files and what skills are associated with his changes. In Figure 6.5(c), we remove the tag for `.java` to show all tags associated with the developer and see what other programming languages he has experience in as well as the projects he has contributed to. Selecting a further tag for a particular skill indicates in which projects the developer has exhibited the skill. Note that tags can also be de-selected in a different order in which they were selected, which supports exploration of the underlying data. Since tags for multiple developers are present in the tag cloud (unless a developer is selected) the tags are sized according to their occurrence in the full list of commits for *all* the developers. When an individual developer is selected the tags will be sized only according to the number of commits in which the selected developer has exhibited the skill.

## 6.5 Comparing Open-Source CVs to Other Developer Profile Data

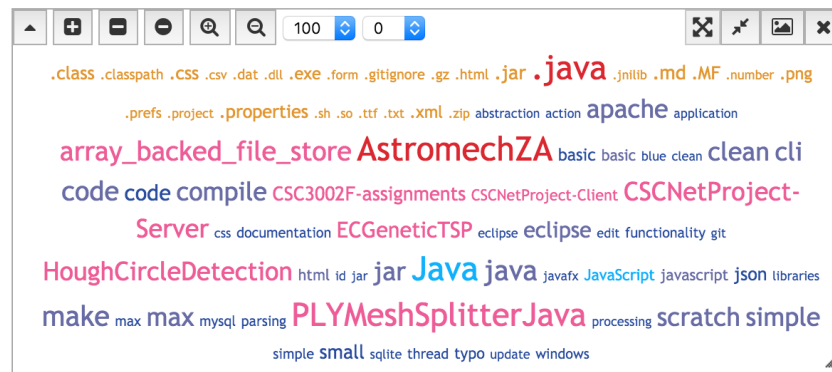
In order to highlight the additional benefit of using skills that can be extracted from GitHub data we compare our open source CVs to developer profiles on Stack Overflow and LinkedIn and show that some skills available on our open source CVs are unlikely to be present on the developer's LinkedIn or Stack Overflow profiles. Figure 6.6 presents a comparison of the types of skills found on each of the three platforms.

### 6.5.1 Comparison to LinkedIn Profile Data

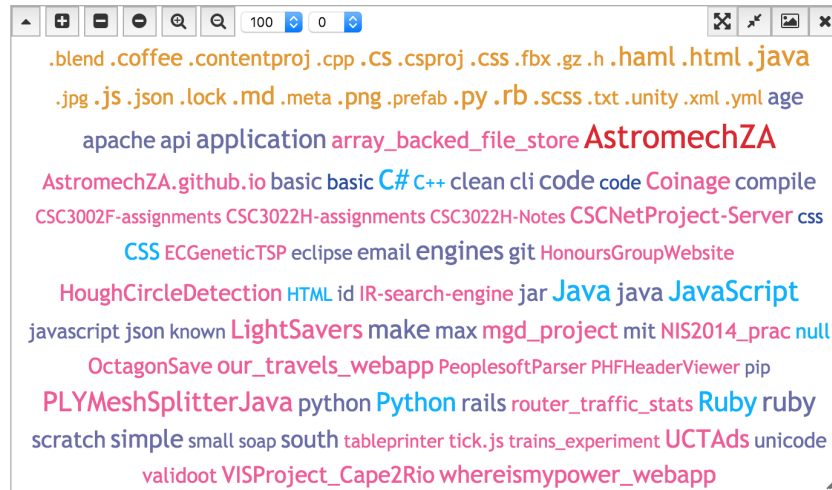
We manually compared the skills represented in our tag clouds to the skills listed on the LinkedIn pages of 30 randomly selected developers listed in our sample from Cape Town. We limited the sample to 30 developers to make the manual comparison feasible. LinkedIn profiles are self-authored and so we manually compare how the self-authored profile differs from extracted skills for the developer. If one set of skills was acknowledged as the ground truth and the other to be evaluated then these comparison metrics are known as precision and recall; however, since neither dataset can be regarded as the ground truth we can only use these metrics in order to compute the overlap and highlight the differences between the two information sources.



(a)

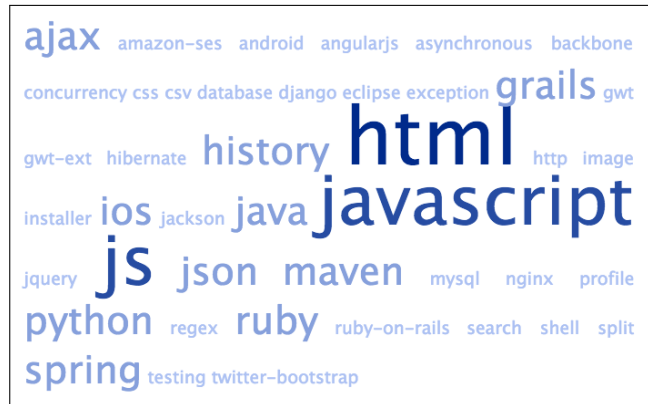


(b)



(c)

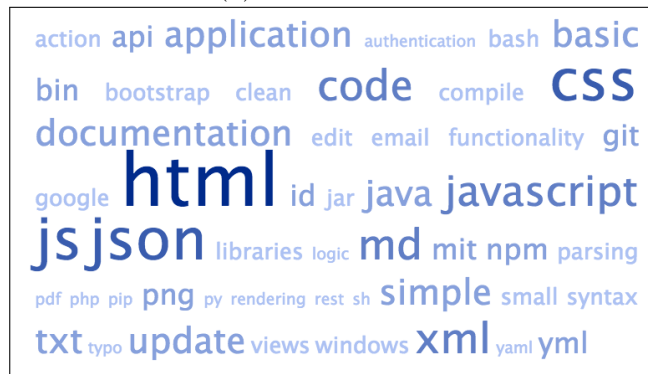
Figure 6.5: Top 100 tags from developers, skills, files and projects associated with a) changing .java files, b) developer selection of AstromechZA editing .java files, and c) developer only selection. Selected tags are indicated in red.



(a) Stack Overflow Skills



(b) LinkedIn Skills



(c) Open-Source CV Skills

Figure 6.6: Tag Clouds of 50 most common skills extracted from 30 developers' (a) Stack Overflow Tags, (b) LinkedIn Skills, (c) Open-Source CVs. Note that tag clouds here have been used purely as a textual visualization to highlight the most frequently occurring skills. The tag clouds have therefore not been generated with ConceptCloud and there is no underlying concept lattice for these clouds. These clouds are therefore presented in a different format.

We manually determined if any of our extracted skills matched those on a user's LinkedIn profile and whether any of the extracted file types or programming languages matched programming languages or web frameworks indicated on the user's LinkedIn profile. Of our sample of 30 developers, six did not have a traceable LinkedIn profile and two did not have any skills listed on their profiles. We extracted an average of 23 LinkedIn skills for the 22 developers that we were able to trace. We extracted an average of 103 skills from the developers' open-source CVs. On average we found that the percentage of extracted open-source skills that also appeared on the developer's LinkedIn profile was 5.1% and the percentage of LinkedIn skills that were also extracted from the developer's open-source profile was 27.3%. This indicates that skill sets extracted from the open-source profiles are complimentary to those listed on LinkedIn profiles. Therefore, examining the data available for a developer on GitHub provides a different view of a developer as this data is unlikely to be available on their LinkedIn profiles. The developers in our sample also tended to list broader skills on their LinkedIn profiles (such as web development and test-driven development) which are too high-level to be matched directly to technical skills on their open source CV.

### 6.5.2 Comparison to Stack Overflow Profile Data

We compared the skills list of the same sample of 30 randomly selected developers to the tags indicated on their Stack Overflow profiles (aggregated from their posts) where we could manually match the developers to their corresponding Stack Overflow profile. We were able to match ten out of the 30 developers to a profile on Stack Overflow.

The ten developers in our sample had an average of 48 tags on their Stack Overflow profiles and 102 extracted skills on their open source CV. An average of 4.8 tags appeared on both profiles. On average 10% of the skills appearing on Stack Overflow could also be found on the developer's open source CV and 4.8% of the skills on their open-source CV could also be found on their Stack Overflow profiles. This indicates that the Stack Overflow skills are also complimentary to those on the open source CV as the same skills are not present on both profiles. The Stack Overflow tags for the developers in our sample also tended to be at a finer granularity and often included particular version numbers (e.g., python 2 and python 3) because they are extracted from posts on Stack Overflow, where they serve to characterize questions in as much detail as possible.

### 6.5.3 Summary

We find that a developer’s LinkedIn and Stack Overflow profiles contain complimentary data to their open-source CV, with little overlap between the profiles. Therefore, examining the GitHub data provides additional insight into a developer’s skill sets as the information obtained is unlikely to be available on Stack Overflow or LinkedIn. We also observe that the skills on all three sites are at a different level of abstraction. Figure 6.6 shows tag clouds of skills of developers in our sample, built from GitHub, LinkedIn skills and Stack Overflow tags respectively; note that numbers and some punctuation have been removed. We observe that there are some common threads through all three data sources, such as high occurrence of Javascript and HTML. We also see that LinkedIn skills are typically higher-level such as “software development” and “web development” and also contain skills such as “Amazon web services” which would be difficult to identify directly using GitHub profiles. However, recruiters indicated that they were interested in a developer’s fine-grained skill set which includes particular web frameworks etc. This information is available from our open source CVs. Stack Overflow tags are more technically detailed and include more general concepts such as “concurrency” which would be difficult to identify from open source contributions. Manual inspection reveals that Stack Overflow tags also commonly contain version numbers of libraries as well, which would in most cases be irrelevant for a recruitment task.

While the data sources show information at different granularities, the lack of overlap suggests that developers omit detail on their self-authored LinkedIn profiles. Therefore it can be beneficial to examine the GitHub data as well. LinkedIn is not developer-specific and therefore developers indicate higher level skills such as “Software Development”, but the lack of detail on their profiles might lead developers to not be identified for jobs or identified for the wrong jobs by recruiters that restrict their search to LinkedIn data.

## 6.6 Identifying Candidates for Open Positions

We evaluate CVExplorer by using it to recommend candidates for two companies and requesting feedback on whether the candidates are suitable to interview. We also asked recruiters for user feedback on the tool.

We have obtained a generic job description for a Software Developer from a large South African company in the financial sector (Company A) and from a smaller company working on social development applications (Company B). We used keywords from their job description in order to match open source developers to these particular positions. We recommended 33 candidates to company A and eleven to

company B. We then asked for feedback on the recommendations as to whether the the companies would interview the candidates or not. The companies were able to evaluate the candidates using both their GitHub and LinkedIn profiles.

### 6.6.1 Suitability of Candidates for Company A

We initially identified twelve candidates for mobile development positions at Company A. One of the candidates that we identified was already in the interview process and had been sourced through other means. All candidates were marked as “typically people we would look to recruit” and “great matches” by a member of the recruitment team. We were then asked by the company to recommend candidates for a further position in a different location (Johannesburg) in South Africa. We recommended a further 21 candidates that had experience in Java, Python, C# or Ruby. The candidates were again marked as suitable to interview, with the C# candidates in particular indicated as exactly the type of developers they were trying to find.

### 6.6.2 Suitability of Candidates for Company B

We identified five candidates for a front-end development position and out of those candidates, two were deemed suitable to interview, one was deemed unsuitable but due to other factors than technical expertise and the CV information for the other two candidates on LinkedIn provided too little information to make a judgment as to their suitability.

We also identified six candidates for an Android development position, out of which two were deemed suitable to interview and the other four were inconclusive because of a lack of information provided on their LinkedIn CVs.

### 6.6.3 Summary

We were able to identify candidates for open positions at two companies and received positive reports from the recruitment representatives of both companies. Successful identification of candidates relied only on selecting a combination of tags that were mentioned in the job specification. In some cases not all tags could be selected at the same time as no single user had all the listed skills but various combinations could be tried to find users that closely matched the job specification. This scenario also indicates the value of an exploratory search approach which includes human involvement so that the search terms can potentially be altered to gather meaningful results. If the list of developers was extracted in report form according to specified tags then the user would not be able to alter the selected skills in order to return the best available selection of developers.



With regards to RQ2 and RQ3 we have demonstrated that an open source CV can be constructed from GitHub data and that CVExplorer can then be used to explore the CVs, allowing successful identification of candidates for positions.

## 6.7 Related Work

### 6.7.1 Mining GitHub User Data for Employment Activities

Hauff and Gousios [89] propose an approach to match job advertisements to developers based on the information available on their GitHub profiles. They use the DBPedia Ontology [60] in order to extract relevant information from job advertisements and from ReadMes of the user's GitHub projects in order to match users to jobs. This work is aimed at developers actively applying for new jobs as opposed to recruiters targeting developers. Therefore, this work also involves indexing of job advertisements in order to match these to developer skill sets which our approach does not. The approach could also be used to automatically match multiple developers to a single job specification. We also process ReadMe files in order to extract skills for developers. However, we do not make use of the DBPedia Ontology so that we obtain the skills in the lowest granularity possible (as they appear in the ReadMe file). While Hauff and Gousios [89] extract only data from ReadMe files, in our approach we are also interested in changed file types (extracted directly from Git repositories), the commit messages and the GitHub project's programming language (extracted from the GitHub API). In our work we are specifically interested in the meaningfulness of the GitHub data in the recruitment context and therefore in addition to utilizing the data we also provide a practical evaluation.

Rusk and Coady [127] extract developers' locations and programming languages to find out which programming languages are used in a particular area. Their tool can also list developers who use a particular programming language in a specified area but no further skills of the developers. Their approach also does not provide any indication of how much expertise a developer has in a particular programming language and does not provide any additional information with which to filter a large developer tool to only relevant developers. The functionality of listing developers in a particular location can also be obtained using the Advanced Search Feature in GitHub [9].

Chen and Sarma [51] have developed a tool, Visual Resume, to compare developers' skill sets in a card format by extracting their contributions on Q&A and open source sites. This is complimentary to our approach as it allows for the detailed comparison of developers' activities across sites, once they have been identified as potential job candidates. We could use this tool as the second step in our pipeline

to provide side-by-side comparisons between developers that have been identified for a particular position.

### 6.7.2 Social Recruiting

Capiluppi et al. [44] discuss how social sites allow for the assessment of developers. They look at advantages and disadvantages of using social sites as a signal for qualifications and references. They observe that the popularity of repositories a user has contributed to can provide a metric for their qualifications and a developer's community can be used as a signal for references. Capiluppi et al. suggest that recruiters should seek help from developers to interpret signals on a developer's code-sharing profile because of the technical nature of the information which could be misinterpreted. We also find that non-technical recruiters are at risk of misinterpreting a developer's GitHub profile because of a barrier to entry and because information on GitHub is not optimally processed in order to aid skills identification.

Vicknair et al. surveyed students across a variety of different fields in 2010 to determine whether they were aware that recruiters were evaluating their social profiles and whether there was data on the profiles that the users did not want recruiters to see [145]. They found that respondents were aware of social recruiting and that they were happy for recruiters to view the content of their social profiles. We found in our survey that developers even expect recruiters to be evaluating their open-source profiles before contacting them.

### 6.7.3 Assessment of Developers by Open Source Developers

Marlow et al. [105] look at how project owners on GitHub assess competency of developers making pull requests to their projects. They found that project owners look at a contributor's other projects, to assess their skill set and to understand their motivation for contributing to the project. The assessment of a developer's skills could have an influence on further interactions, such as providing more gentle commentary to a developer who is new to GitHub when rejecting the developer's pull request. CVExplorer can also be used by developers to easily view an aggregation of a GitHub user's skill set.

Dabbish et al. [58] report that the recency and volume of a developer's contribution was used as an indication of the developer's commitment to open source development. In addition, a developer's activity over all projects was used to determine what the developer was currently interested in, which could also be obtained via CVExplorer.

Singer et al. [133] investigate the use of profile aggregators by developers and recruiters. They find that developers are assessing each other and are also consciously

managing their own profiles. Recruiters were also surveyed on their strategies for connecting with developers, with some successfully using profile aggregators and some of them experiencing too many barriers to entry. Recruiters also mentioned that they try to filter developers by skills, which is in line with what we have gathered from recruiters in our own survey.

## 6.8 Conclusions

We surveyed 85 open source developers in order to establish the role of GitHub in identifying (sourcing) technical candidates. We also surveyed 7 and informally interviewed 6 recruiters to complement the developer perspective. Our findings from the recruiter survey corroborate sentiments expressed in a previous survey of 13 recruiters [133]. Developer responses indicated that recruiters have difficulty interpreting their GitHub profiles. Developers also report receiving generic recruiter emails, which cause them frustration, even from recruiters that have evaluated their GitHub profiles.

We have mined developers' skill sets to form open-source CVs and made this information browsable in a CVExplorer. Our approach supports recruiters in identifying relevant candidates using the GitHub dataset which can prevent developers from receiving ill-fitting job offers from recruiters. We have used CVExplorer to recommend candidates for open positions at two different companies, where one of the companies has then requested recommendations for a further position. We have also compared the granularity of technical skills data extracted from GitHub projects to that of skills data available on LinkedIn and Stack Overflow and observed that the data sources are complementary.

The identification of candidates using GitHub data is based on their portfolio meta-data as opposed to a self-authored CV which may be inaccurate or fail to list skills in sufficient detail. Since GitHub profiles are generated from the user's activity they always contain up-to-date information about the developer's exhibited skills. More accurate identification of passive candidates can increase the quality of hiring decisions which can therefore decrease the risk that poor hiring decisions pose to the success of a software project [138].

## Chapter 7

# Concept-Based Exploration of Academic Publication and Citation Data

In this chapter, we describe how we have applied the ConceptCloud browser framework to a collection of academic publication data. The application to academic publication data demonstrates the flexibility of our approach. The ConceptCloud publication browser can be used to explore publications in a large dataset and to familiarize new researchers with a field. The browser displays aggregated reference and citation data in the tag cloud view to allow for the easy identification of trends within the references and citations. Our publication browser also demonstrates the scalability of our approach where the underlying context table contains over 1.8 million objects (academic papers). The handling of reference and citation data also requires us to develop a new mechanism for handling relationships between objects in our approach which the underlying concept lattices do not directly support.

### 7.1 Introduction

There are a large number of academic papers available for a variety of different topics and new research and fields are being added continuously which makes it difficult for researchers to keep up to date with a field or to find relevant related work [114]. Therefore, mechanisms for researchers to discover papers of interest (for example, Mendeley's [92] or Google Scholar's [55] paper recommendations) are becoming increasingly relevant. While automatic paper suggestions can be used to highlight possibly relevant work for researchers, they make use of a researcher's academic profile and therefore cannot support novice researchers, or those unfamiliar

to a topic, in finding the most relevant work on new topics. Traditional approaches for finding relevant academic publications are search-based and rely on users to manually specify keywords of interest.

We present the 2014 ACM Digital library data in the ConceptCloud browser in order to allow the publication data to be queried flexibly and to support exploratory search tasks. Since the ACM data contains titles and abstract data for the publications, we perform our own key-phrase extraction (see Section 7.2.2) in order to generate normalized terms that can be used for finding work on a specific topic.

Citation and reference information can also be used to find new papers on a specific topic or highlight the foundational work for a specific paper. However, aggregated information about citations for a specific topic or author are difficult to obtain from traditional list-based interfaces commonly presented by digital libraries. We also use the reference and citation data to enable users to follow forward and backward links to additional research that may be relevant to a topic. In our approach we allow users to view a tag cloud of reference and citation information for a single paper presented in an aggregated form which is novel. For example, the user can quickly get a summary of the papers that have been referenced by or cited a specific paper and easily see whether there are particular authors that have repeatedly cited the paper or particular conferences or journals which have often cited the paper. We also allow users to view aggregated citation and reference data for a group of papers that all have a selected property. For example, users can view aggregated citation or reference information of all papers by a certain author, all papers in a particular conference or all papers on a certain topic. Figure 7.1 shows an example of our tag cloud browser which shows the tags associated to the selected key-phrase (model checking) as well as the tag clouds of papers that have been referenced by “model checking” papers (bottom left) and papers that have cited “model checking” papers (bottom right).

Academic paper collections consist of semi-structured data where the paper itself is free-text but there are structured fields such as the venue and year of publication which also provide information about the paper. Using only the structured fields to support search tasks on the paper collection would not be optimal as the topical information contained in the paper’s text would then not be considered.

We conducted a user study (see Section 7.6) in order to evaluate the usability of the ConceptCloud publication browser. We asked participants to answer a variety of scientometric questions using ConceptCloud and timed them for each question. Our user study showed that participants were able to answer complex scientometric questions using our interactive tag cloud interface with an average accuracy of 73%.

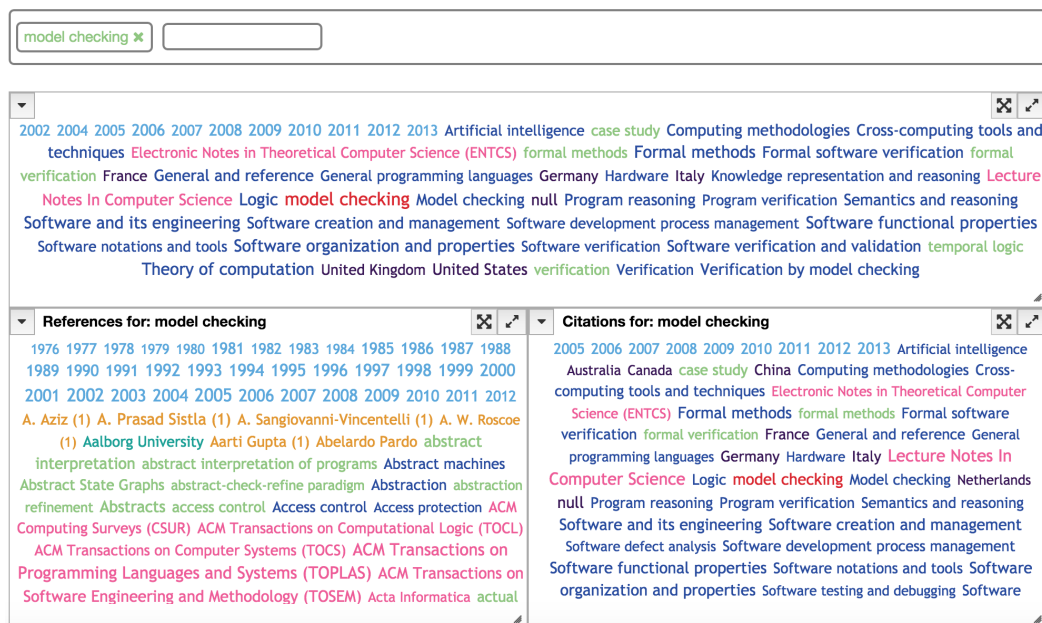


Figure 7.1: Three tag clouds with a selected tag (red), authors (orange), years (cyan), and keyphrases (green). For each tag cloud the 100 largest tags are shown from papers associated with the keyphrase Model Checking (top), papers that are referenced by papers associated with the keyphrase Model Checking (bottom left), and papers that cite papers associated with the keyphrase Model Checking.

## 7.2 Constructing Contexts from Publication Data

### 7.2.1 Paper-Based Contexts

We construct contexts using the academic paper as the object in the context, represented by its unique ID, and additional information about the papers as attributes. For each paper we have the authors, which are disambiguated according to the ACM Digital Library dataset, author affiliations, the paper title, abstract as well as the ACM classification for the paper and the publication series and year. Not all information is available for every paper in the dataset. In order to use the abstract and title data in the context table we first extract key-phrases from these fields. Figure 7.2 presents an overview of our approach to publication browsing. We initially extract all the relevant metadata from a set of publications. We then use this metadata to construct a concept lattice and resulting tag cloud.

Note that there are some drawbacks of using the papers as the objects in the context table as in this way additional information about the authors, such as their affiliations, is assigned to the papers and not directly to the authors. However, while this information can then not be used to identify the institution of the authors themselves it can still be used to identify collaboration between authors of different

institutions and the institutions of the authors contributing to a particular paper. Using the authors as the objects in the context table would cause us to lose more fine-grained information about the individual papers (i.e., we would have fewer objects in the context table and more attributes associated to each of the objects) and would only have the benefit of allowing us to identify the institutions of a particular author. Since the goal of this application is to facilitate the browsing of academic papers and their topics, and allow users to identify relevant academic papers we have chosen the paper itself as the most suitable object in the context.

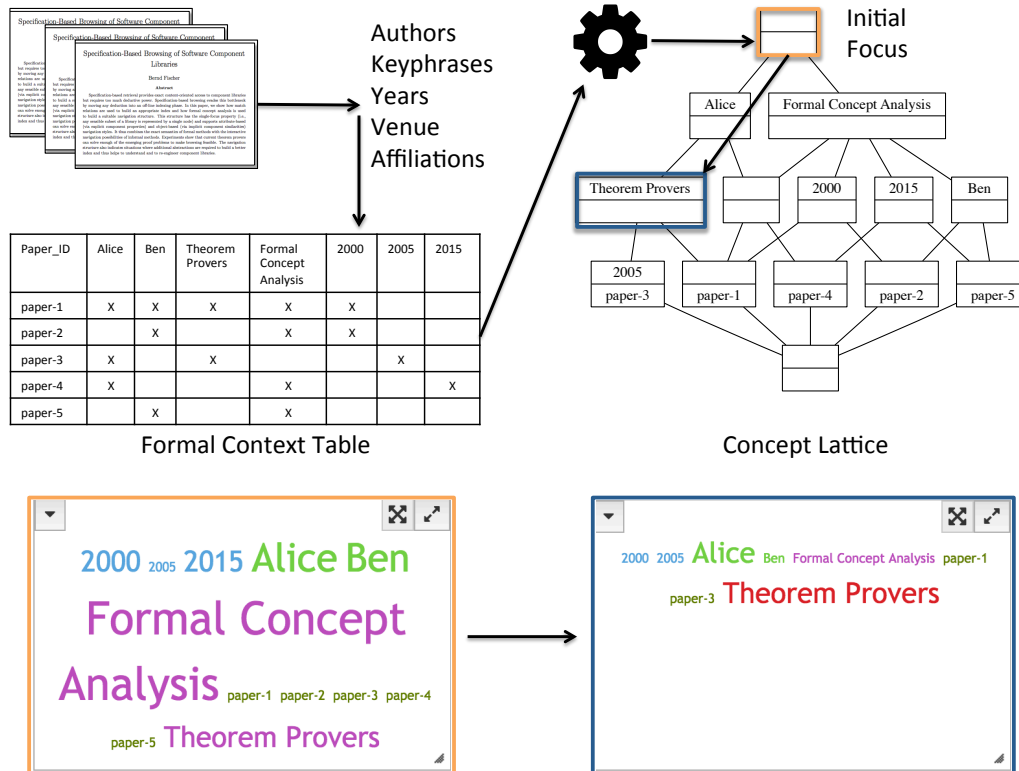


Figure 7.2: Overview of the process of creating a tag cloud from publication data. We extract key-phrases directly from the titles and abstracts of each paper. Our context tables are paper-based and we associate all of the paper's attributes (e.g., key-phrases, author names, author institutions, publication venue and year) to the paper in the table. The concept lattice is then constructed directly from the context table and the information is made available for exploration in an interactive tag cloud.

### 7.2.2 Key-Phrase Extraction

We follow the key-phrase extraction technique outlined in [66]. We use an unsupervised key-phrase extraction technique as there is a lack of suitable training and

test data (which would need to be manually provided for each academic paper) for academic publications. We perform stemming and lemmatizing on the words in the key-phrases. Since the stem is not always a proper word we select a representative for each stem that is most common across the whole dataset and not only in the individual paper so that we avoid having multiple similar key-phrases in the tag cloud. Lemmatization itself is not sufficient because similar words representing different parts of speech are not grouped together which would therefore result in a tag cloud that contains duplicate tags which have essentially the same meaning.

### 7.3 ConceptCloud Publication Browser

The tag cloud of publications includes author names, key-phrases, year of publication, series of publication, ACM classification concepts, author affiliations and countries. Note that as papers are used as the objects in the context table, multiple authors along with their affiliations will be assigned to a single paper. Therefore when an author tag is selected the affiliations (and therefore also countries which are derived from the affiliations) shown in the tag cloud will include entries for all authors of all papers the selected author has worked on. Therefore the affiliation information provided in the tag cloud can easily be used to identify collaborations (i.e., between authors, institutions and countries) but the data cannot be used to indicate which affiliation a particular author has (as the affiliations for the author's collaborators will be shown in the tag cloud as well). However, if we were to use authors as objects then this would no longer be the case, but we would then not be able to get information for individual academic papers as all paper information would be assigned as attributes to the authors.

Since the publication dataset is prohibitively large we limit the initial tag cloud to showing only the largest 5000 tags from the full tag cloud. We therefore provide a search bar in the browser so that all tags are still available for selection through the search bar. The search bar itself only allows users to select tags that are present in the full tag cloud. Figure 7.3 shows the initial view of the ConceptCloud publication browser. Each subsequent view in the tag cloud after an initial selection is restricted to showing only 5000 tags so as not to present too much information to the user. Therefore the tag cloud view will be incomplete until the user has made sufficient refinements to yield a tag cloud of less than 5000 tags. Additionally each viewer provides an option to sort the tag cloud by count; this enables users to easily identify for example, the author with the most papers on a specific topic.

We have also added a table view to the publication browser, which shows the titles of the papers that correspond to the current tag selections and orders these



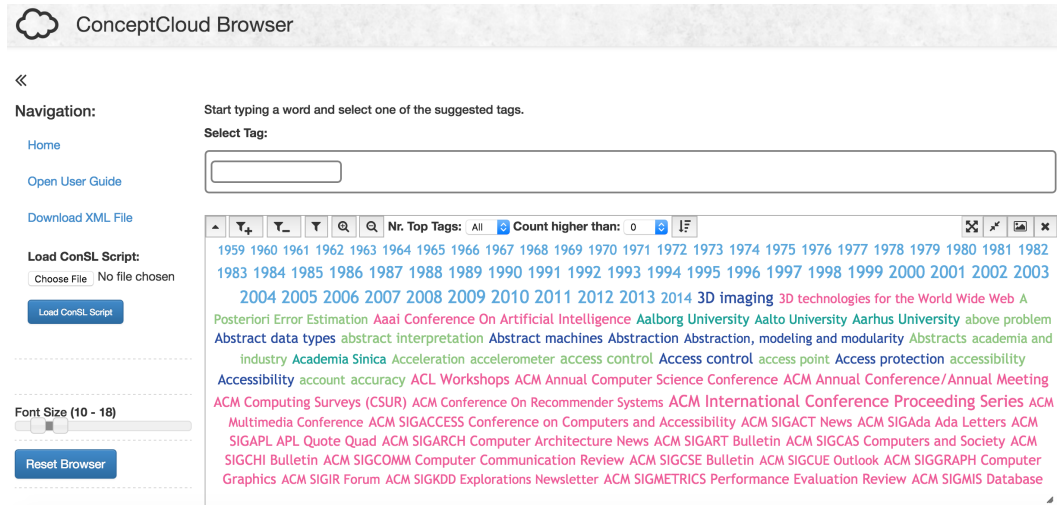


Figure 7.3: Initial view of the ConceptCloud Publication Browser. The main tag cloud presents the 5000 largest tags in the tag cloud. Other selections can be made using the search bar provided.

by citation count. Figure 7.4 shows an example of the table view which presents additional information for each paper. The table view enables users to see which individual papers correspond to the tag selections that have been made in the tag cloud and allows users to retrieve the full text of the papers using the full title which is not provided as a tag in the tag cloud.

Title	Year	Abstract	Series	Authors	Key Phrases	Concepts	Citation Count
Distinctive Image Features from Scale-Invariant Keypoints	2004	This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. ...	International Journal of Computer Vision	David G. Lowe (1)	object recognition, fast nearest-neighbor algorithm, reliable matching, scale invariance, least-squares solution for consistent, image matching, invariant features	Computer vision, Computer vision problems, Shape representations, Computer vision representations, Computing methodologies, Object recognition, Computer graphics, Image manipulation, Artificial intelligence	3729
Fast Algorithms for Mining Association Rules in Large Databases	1994		Very Large Data Bases	Ramakrishnan Srikant (1), Rakesh Agrawal (1)	mining association rules, fast algorithm, large databases		3069

Figure 7.4: Table view of publications in the ConceptCloud Publication Browser

## 7.4 Handling Object References with Formal Concept Lattices

The publication dataset contains relationships between objects, as papers can cite each other. There is no direct way to support relationships between two objects directly in the concept lattice. Using a database structure we could make use of a bridge table which would allow us to record the reference and citation relationships in a bridge table that links papers to each other. However, in a concept lattice structure, objects can only be linked by a shared attribute, and there is no natural mechanism to indicate that two objects are related to each other.

The most obvious solution for relating different objects in the context table is to add references to other objects to the attribute set of a single object. In that way academic papers would be used as both objects and attributes (i.e., whenever they are referenced or cited) in the context table. The category of the attribute could be used to indicate whether the relationship between the two papers is a reference (i.e., an outgoing link) or citation (i.e., an incoming link). However, following this approach selecting a paper in the tag cloud would show only tags for the papers (represented by their paper-ids) that form references and citations for the selected paper. No additional information about the references and citations (such as authors, keyphrases or publication years) would be present in the tag cloud and it would be necessary to select the tag (the paper-id) for each individual referenced and citing paper to gather any information about the papers themselves. Therefore, while using referenced and citing papers as attributes in the context table is possible, this does not provide any additional information about the referenced and citing papers and is limited in usefulness in terms of the navigation options provided.

In our approach we make use of multiple concept lattices in order to represent the reference and citation data for a publication or a set of publications. We have one main concept lattice which contains the data for all publications in the dataset. However, for each paper  $a$  we also have a smaller concept lattice which contains all other papers that cite  $a$ . Therefore the citations to paper  $a$  can be explored by exploring the smaller concept lattice which contains only the citing papers of  $a$ . In practice the generation of all citing and reference concept lattice would be impractically inefficient. We therefore generate reference and citing concept lattices on-the-fly as they are needed (created by the user).

Note that while there has been work on handling complex objects [72] and concept references [93], in our approach we are concerned with using the data to facilitate intuitive exploration through the tag cloud interface.

Any tag in the tag cloud can be used to construct an additional concept lattice

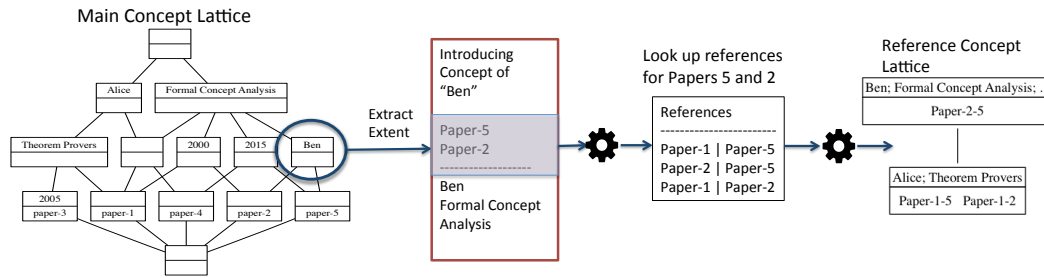


Figure 7.5: Overview of reference context creation for author **Ben**. We use the extent of the attribute concept for the author **Ben** to look up the references for his papers. We then use these papers to create a new context table whose object set is a subset of the objects in the full context table.

of the references or citations for a particular attribute (which represents a set of publications) in the dataset. Therefore, it is possible to generate a tag cloud from the newly-created concept lattice which displays aggregated information about the reference or citations for an individual paper, but it is also possible to generate a tag cloud which displays aggregated information of papers that have cited a particular author, or papers that have cited papers that apply to a particular key-phrase (i.e., a group of papers that have a particular key-phrase or author).

Figure 7.5 shows an overview of the process of creating a reference cloud for the key-phrase **Ben**. We first find the extent of the introducing concept for **Ben** and then look up the references for all papers in the extent of the concept (here **paper-2** and **paper-5**). We then generate a new concept lattice using as objects all the papers referenced by **paper 2** and **paper 5** (in this example, **paper-1** and **paper-2**). Note that the new object type uses a combination of the paper-id and the reference paper-id; in this example **paper-1** is referenced twice, once by **paper-2** and once by **paper-5**. Therefore in the resulting tag cloud, attributes of **paper-1** will carry a higher weight (larger tag size) than the attributes of **paper-2** as **paper-1** has been referenced in more than one of **Ben**'s papers. Therefore, the object types are no longer simply the paper IDs as the number of citations and references are included in the context table as well.

When an additional concept lattice of references or citations is created, we present a new tag cloud corresponding to the newly created concept lattice in the browser and allow users to navigate in this tag cloud as well, in order to browse the references and citations of a paper or attribute. For example, Figure 7.1 shows the tag cloud for key-phrase **model-checking** as well as the tag cloud for papers that are referenced by papers with the key-phrase **model checking** (bottom). Both of these tag clouds are available to filter and navigate in.

When a selection is made in any of the tag clouds it will be applied to all clouds

that are currently visible on the interface. However, since we have introduced tag clouds with different underlying concept lattices selections can only be applied if a tag for the selection exists in the underlying concept lattice. Therefore, where the selection is available in all tag clouds all tag clouds will be updated with the new selection and where the selection does not exist in any of the tag clouds the tag cloud will not be updated.

## 7.5 Worked Examples

We show what types of information the ConceptCloud publication browser can be used to identify in this section. The tag clouds can be used to identify the prominent authors in a particular field as well as collaboration between different authors and institutions. Examining the topics published at different venues also provides an overview of the topic trends at a particular venue. Additionally the tag clouds of reference and citation data can be used to answer questions such “Who is citing my papers?” and “What kind of papers cite this paper?”.

### 7.5.1 Prominent Authors

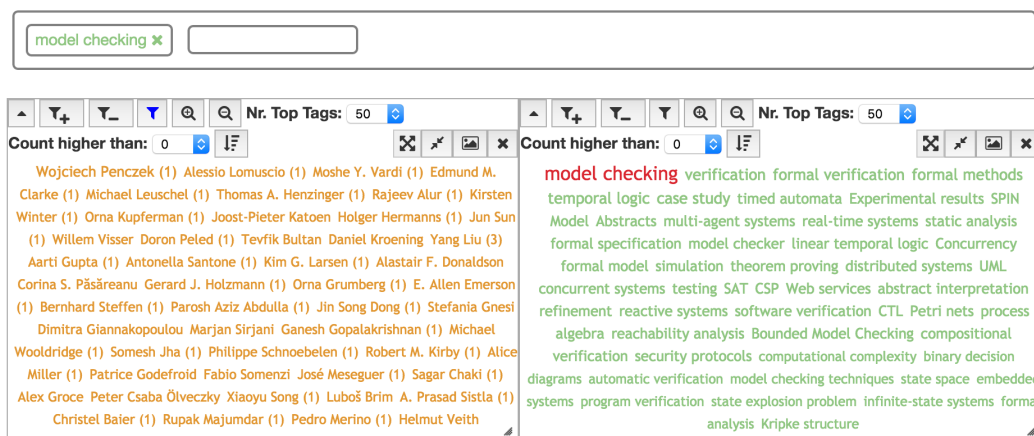


Figure 7.6: Top 25 prominent authors on Model Checking sorted according to the number of publications on Model Checking (left). Top 25 key-phrases of papers that have Model Checking as a key-phrase (right).

In order to identify the prominent authors in a specific topic, there are a variety of metrics that can be used. For example, we can look at the number of publications someone has written on a topic, the authors of the most cited paper on a particular topic or the author that has been referenced the most by work on a specific topic.

Using our publication browser we can examine the prominent authors in a variety of ways.

For example, to evaluate the prominent authors on **Model Checking** we can look at which authors have the most papers with the key-phrase **Model Checking**. In ConceptCloud we only need to search for **Model Checking** and restrict the view to showing only the author tags. Figure 7.6 shows the tag cloud for the top 25 authors with papers where **Model Checking** is one of the key-phrases, sorted according to the number of such publications. Additionally we can also see what other key-phrases most commonly occur along with **Model Checking**. In our dataset author **Wojchciech Penczek** has 28 papers on **Model Checking**. However, from the table view we see that the most-cited **Model Checking** paper in our dataset is authored by **Corina S. Pasareanu**, **Sarfraz Khurshid** and **Willem Visser** and so these authors could also be considered prominent authors.

### 7.5.2 Author Collaboration

In order to identify author collaboration in the tag cloud we can select the tag for any author and see which authors they have collaborated with, sized according to how often they have collaborated. Figure 7.7 shows as an example which authors **Robert Sedgewick** has collaborated with, sorted according to the number of papers they have collaborated on.

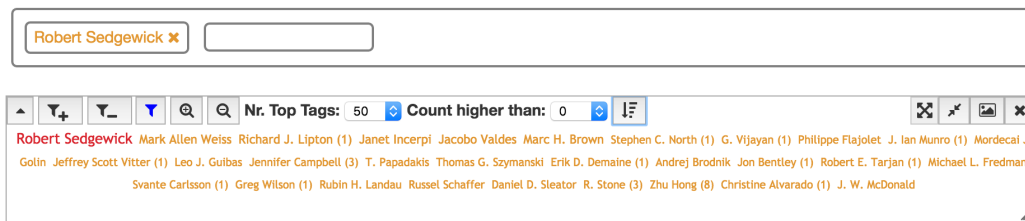


Figure 7.7: Author collaboration cloud of **Robert Sedgewick**. Tags are ordered according to number of papers the author has written where **Robert Sedgewick** is also an author.

### 7.5.3 Topic Trends

In order to look at the trends in topics in a specific conference or journal we can select the tag for a conference or journal and then create “sticky tag” viewers that allow us to compare the topics in different years. For example Figure 7.8 shows an example of the key-phrases from papers in the **International Conference on Software Engineering** over two different years (2000 and 2013 which is the last complete year

of publications contained in our version of the ACM DL dataset). The sticky tag viewers allow direct comparison of the key-phases that were contained in papers of the two different years of the conferences.

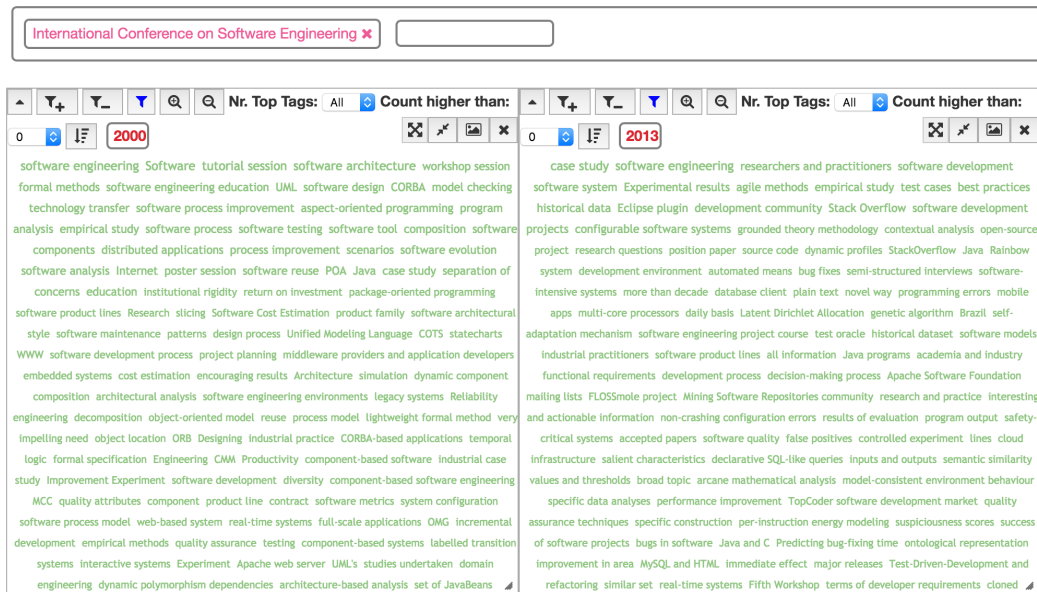


Figure 7.8: Key-Phrases of papers in International Conference on Software Engineering for the years 2010 and 2013 presented in a sticky tag viewer to enable easy comparison. The tag clouds are sorted to show the key-phrases that apply to the most publications first.

#### 7.5.4 Citation Analysis

The reference and citation clouds allow us to explore a large amount of different information pertaining to citations and references. For example to see the referenced work for a particular topic, we can create a reference cloud for the topic and examine the key-phrases of the referenced papers. Figure 7.1 displays the reference and citation clouds for the key-phrase *model checking*.

### 7.6 User Study

We performed a user study to evaluate whether users are able to answer complex scientometric questions through our iterative tag cloud interface. Participants were asked a wide variety of scientometric questions and evaluated on whether they could find and retrieve the correct information, such as papers related to a particular topic of interest, prominent researchers in a field, and journals and conferences best suited

to different research topics. For this user study we used the ACM DL dataset to populate the publication browser which consists of 1 848 048 papers of which 1 399 805 have abstracts that are necessary for the keyphrase extraction. The user study was set up primarily to evaluate the usability of the publication browser. Secondly, in context of the ACM DL dataset, the scientometric questions served to test whether the participants were able to use the publication browser’s functionalities to retrieve the correct information and whether they interpret the visualized information correctly.

Note that since we used the ACM DL dataset to populate the publication browser, it only contains citation and reference information from the closed citation network of the ACM DL. Therefore, the citation counts displayed in our browser differ from other indexing services and the correctness of the participants’ answers in the user study are judged strictly in the context of the ACM DL data. Our publication dataset is restricted to that of the ACM DL because of the unavailability of other publication datasets that include citation information and full abstracts of papers. However, our browser is flexible enough to be used with any other academic publication datasets and new data sources could simply be added into the context table by merging paper and author entities, should new data sources become available in future.

For the purposes of the user study we grouped the browser’s functionalities into eight main categories, such as selecting tags and creating viewers, and formulated the questions so that participants are required to use all of the tool’s functionalities to answer the questions successfully. The list of function categories with the number of questions requiring the corresponding functionalities is given in Table 7.1. For each question, we could identify which tool functionalities were needed to answer the question and which functionalities were used by the participants.

Table 7.1: Main functionalities provided by the publication browser grouped into categories. The second column shows the number of questions in the user study that require the associated browser functionality to answer the question.

	<b>Functionality</b>	<b>Nr. of Questions</b>
F1	Selecting/Deselecting Tags	17
F2	Searching for tags using the provided search functionality	12
F3	Filtering categories and creating category-specific viewers	18
F4	Filtering viewers by number or size of tags	15
F5	Sorting the tags according to tag count	15
F6	Creating viewers with “Sticky Tags”	2
F7	Creating Citation and Reference clouds	4
F8	Interpreting the table view	3

The number of questions associated with the various functionality categories differ substantially. This is due to the fact that certain functions are prerequisites for others. For example, selecting or searching for a tag is required before creating a citation or reference cloud. Rather than forcing an equal number of questions for each functionality, the distribution more closely reflects a real-world use case.

Similar to the study conducted by Osborne et al. [112], we performed a non-comparative user study since most academic indexing platforms and digital libraries do not provide the required functionality to answer the type of complex questions that our publication browser has been designed to answer. Therefore, a comparative study between two or more tools would unfairly favor one tool over the others depending on the questions constructed. If we constructed an equal number of questions that could be easily answered with each of the tools in the study then the results of our study are highly likely to be inconclusive.

Participants first watched an introductory video (available at <https://www.youtube.com/watch?v=8zJ618y0WBI>) about the tool to familiarize them with the usage of the tool. Each participant session was conducted separately and we observed each participant and made notes of exactly which tool functionalities they made use of and how long they took to answer each question. We used a stopwatch to time the participants on each question and did not include the time they took to write their answers on paper as the participants writing speeds would have varied largely. We applied a total time limit of 90 minutes per participant to complete the study, in order to have an end time that we could provide to participants for their scheduling purposes. We checked that answering all questions was feasible with this time limit by answering the questions ourselves, and asking a new user (not part of the study) to answer the question set and timing them.

### 7.6.1 Question Set

We constructed 22 questions that cover various topics in scientometrics to establish whether users are able to answer common scientometric questions using the publication browser. The questions cover eight broadly defined topics such as citation analysis [61, 78], author collaboration [33, 146], author ranking [91, 149], topic trends [126], and university rankings [35, 68]. The complete list of topics covered by the questions is given in Table 7.2.

Some of the questions in our study are formulated more generally to evaluate how the participants would make use of the tool in order to answer scientometric questions according to their own interpretation of the question. For example, finding a prominent author depends on the participant's interpretation of whether citation counts or the number of publications are used as a metric to define "prominent".



Table 7.2: The number of questions for each topic proposed to participants in the user study.

	<b>Topic</b>	<b>Number of questions</b>
T1	Citation analysis	6
T2	Author ranking	2
T3	Author activity	4
T4	Author collaboration	5
T5	Topic trends	3
T6	University ranking	2

We have phrased question Q1 as “Who is the most prominent author in the field of Machine Learning?” rather than directly asking participants to select the tag for the keyphrase “Machine Learning” and evaluate which author has the largest tag size in the tag cloud (most publications on this topic) as that would only measure the participant’s ability to select tags and follow instructions and not their ability to answer scientometric questions with the tool. As a result of the question phrasing, there are multiple “correct” answers for some of the questions, and during the study we recorded how the participants made use of the tool to answer the questions.

Table 7.9 lists the questions that the participants had to answer during the user study. The table indicates the category in which each question falls (T1-T6), as well as the tool functionalities (F1-F8) that need to be used to answer the question successfully. However, in the user study the participants were only provided with the questions without any additional hints, such as the topic of the question or the required tool functionalities. We also randomized the order of the questions for each participant to allow us to identify any learning effects during the course of the participants’ usage of the tool. Note that the names of authors, series and years are arbitrarily selected, and replacing these variables with other values would still require participants to answer the questions in the same way.

### 7.6.2 Population

Our participants were post-graduate students and post-doctoral researchers from various departments at Stellenbosch University. Participation in the study was voluntary and participants were not compensated in any way. A total of 39 participants from various academic fields took part in the user study.

We first performed a pilot user study with five participants (see Section 7.6.3), after which we made changes to the search functionality on the browser and the question set. The results shown in this chapter are obtained from conducting the user study with the remaining 34 participants. Figure 7.10a shows the distribution of postgraduate research experience in years for the 34 participants, while Figure 7.10b

Figure 7.9: User study questions along with the tool functions required to answer each question and the topic of the question. Optional functionalities are indicated in brackets and interchangeable functionalities are indicated with a slash.

	<b>Question</b>	<b>Functions</b>	<b>Topic</b>
Q1	Who is the most prominent author in field of “Machine Learning”?	F1/F2, F3, F4/F5	T2
Q2(a)	Which country does “Germany” collaborate with the most?	F1/F2, F3, F4/F5	T4
Q2(b)	In which year did authors from these countries publish the first collaboration paper?	F1, (F3)	T4
Q2(c)	How many authors collaborated in this year from these two countries?	F1, F3	T4
Q3	Which are the top 5 affiliations according to publication counts in 2010?	F1/F2, F3, F4/F5	T6
Q4(a)	List the three main topics author “Ivan Bratko” works on.	F1/F2, F3, F4/F5	T3
Q4(b)	Create a viewer showing the years he published papers about his most prominent research topic. Show the viewer to the observer of the study.	F1/F2, F3	T3
Q4(c)	In which years did he publish the most papers overall?	F1, F4/F5	T3
Q4(d)	With whom does he co-author the most papers?	F3, F4/F5	T4
Q4(e)	Name the author who cites him the most.	F3, F4/F5, F7	T4
Q5	What are the 3 main keyphrases of papers that cite papers with the keyphrase “Machine Learning”?	F1/F2, F3, F4/F5, F7	T1
Q6(a)	In what year were the first papers published about “Internet of Things”?	F1/F2, (F3)	T5
Q6(b)	If you had a paper about “Internet of Things”, in which journal/conference would you publish if you’re only interested in being cited as often as possible?	F3, F4/F5, F7	T1
Q7(a)	Compare the topics in the first and last year of the series “International World Wide Web Conference”. How have they changed over time?	F1/F2, F3, F4/F5, F6	T5
Q7(b)	What is the title of the paper with the most citations in this conference?	F8	T1
Q7(c)	Who published the most papers in the first year of this conference?	F1, F3, F4/F5	T2
Q7(d)	Where did this author publish most papers?	F1, F3, F4/F5	T3
Q7(e)	In which journal/conference is this author’s most cited paper?	F1, F2, F8	T1
Q8	What are the most prominent keyphrases covered by the “International Conference on Software Engineering” and how do these differ from the conference “Foundations of Software Engineering”?	F1/F2, F3, F4/F5, F6	T5
Q9(a)	Which is the most cited “International Conference on Software Engineering” paper and what is its topic?	F1/F2, F8	T1
Q9(b)	Which journal/conference is most referenced in “International Conference on Software Engineering”?	F3, F4/F5, F7	T1
Q10	If you had to advise “Ben Carterette” on choosing the most relevant university to collaborate with, which university would you suggest and why?	F1, F2, F3, F4/F5	T6

indicates their academic ranks.

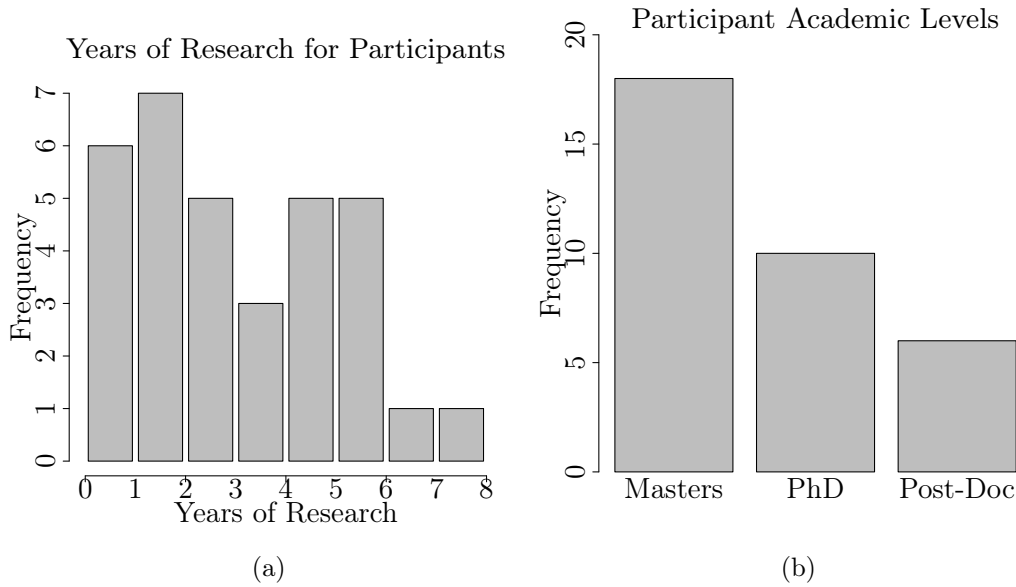


Figure 7.10: Population characteristics. (a) Years of research experience of participants. (b) Participants' academic ranks.

No formal ethical clearance was sought for this experiment because this was a minimal risk study but ethical considerations were taken into account when designing the study. No personal data was collected or stored for the participants and the question answers were stored on a password protected computer. Verbal consent was obtained from the participants after the study set up and goals were explained to them. Participation in the study was voluntary for all participants. Collected data will be destroyed at the end of the research project.

### 7.6.3 Pilot Study

Initially, we performed a pilot study with five participants after which we re-evaluated the user study setup before continuing. The tool initially included a traditional search interface which allowed users to search for tags by submitting free-text queries. In this application the search functionality was particularly important because the tag clouds were restricted to showing only 5000 tags. Therefore, it was often necessary for participants to select tags that were not currently shown in the tag cloud and they then needed to make use of the search functionality. In addition, all questions were initially formulated to be self-contained without any continuation from one question to another. The traditional search interface and the structuring of the questions led users to approach the tool as a traditional search tool because of the familiarity with

Start typing a word and select one of the suggested tags.

Select Tag:

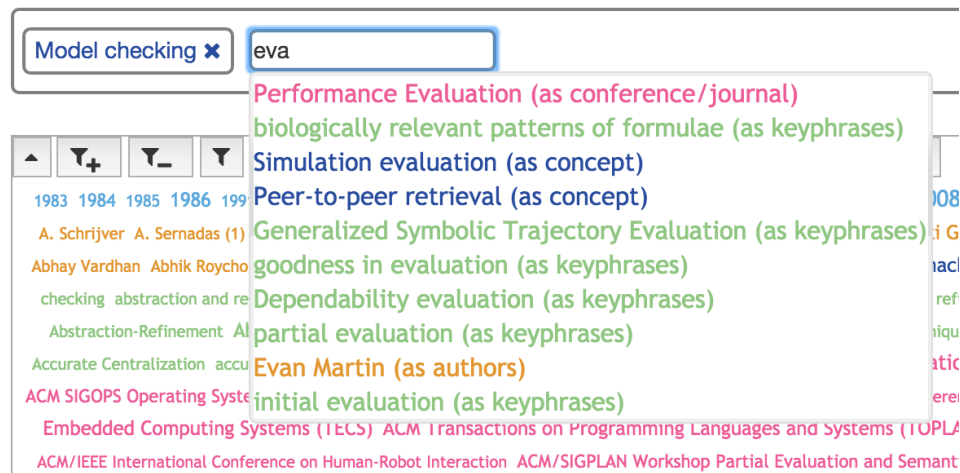


Figure 7.11: An example of the search functionality provided by ConceptCloud. Tags can only be selected from the suggestions that appear when the user starts typing a particular word. No free text searches are possible through this interface to encourage browsing. When one tag has been selected suggestions will only be displayed for tags that can be selected in the current tag cloud (i.e., where the tag can be selected in conjunction with the currently selected tag and will cause an empty tag cloud to be displayed).

the concept of search and they therefore did not approach the tasks in an exploratory manner.

We adapted the question set to include questions that continued on from a previous question, which would force users to adopt a browsing and exploratory search [151] approach to answer the questions efficiently. For example, we changed the follow-up question of “List the three main topics author “Ivan Bratko” works on”, from “In which years did “Ivan Bratko” publish most papers overall” to “In which years did he publish the most papers overall”. We also added additional follow-up questions. For example, we added questions Q2(b) and Q2(a) to question Q2 (see Table 7.9), to encourage users to keep current tag selections and to browse the publication data instead of using the search functionality at the beginning of each question.

We also changed the search interface so that participants could no longer search using free-text queries but instead had to select a tag from a list of suggestions after they started typing, as shown in Figure 7.11. Furthermore, using the associated color for the suggestions according to the color of the tag categories in the tag cloud would make it more obvious that the search function was simply another mechanism of selecting a tag, instead of manually looking for a specific tag in the tag cloud.

In the pilot study we provided participants with only a user manual detailing the tool’s features. However, since we noticed that not all participants observed the manual, we showed an introductory video describing the use and functionalities of the tool to the participants of the main user study before beginning each individual session.

## 7.6.4 Results

In the user study we collected detailed information about each participant’s usage of the tool. For each question we recorded the time taken to complete the question, the functionalities that the participant used to answer the question, as well as, the correctness of the answer according to the participant’s interpretation of the question. Note that each participant received the questions in a random order which enabled us to evaluate learning effects.

### 7.6.4.1 Time Taken per Question

To evaluate whether there was an observable learning effect for participants we plot the average time taken for questions in question order (Figure 7.12a) and the average time taken for questions in the order that the participants received them (Figure 7.12b). For example, Figure 7.12b shows the average time taken by the participants to answer their first questions despite the fact that the actual question they were answering was random. Figure 7.12c shows the normalized time for each participants’ questions in the order that they received them. The normalized time for participant A’s first question (Q7) is the time they required to answer Q7 minus the average time that all participants required for answering Q7.

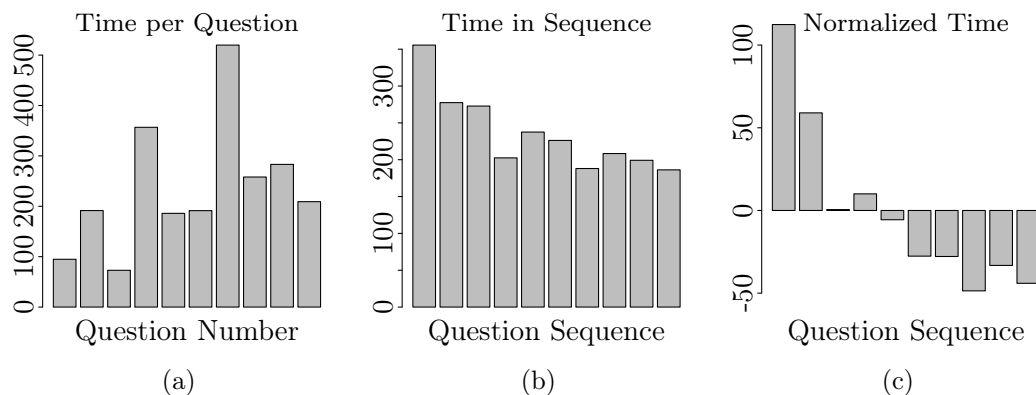


Figure 7.12: Average time taken per question in (a) question order (b) the order that the participants received them (c) the order that the participants received them normalized by the average time of all participants per question.

A comparison between Figures 7.12a and 7.12b shows that the patterns of question timings differ substantially when the average time was calculated according to question number or question sequence. From 7.12a one can see that participants took the longest time to answer question Q7 followed by Q4, both of which comprise of multiple subquestions. Figure 7.12b shows that, on average, participants took the longest to answer their first question, regardless of what question this was, and the time decreased as participants answered more questions. Figure 7.12c confirms this learning effect and shows that when a participant started their first question they took more than 100 seconds longer to answer it than the average time needed on that question by all participants. By the third question the participants were able to answer it as fast as the average time required for that particular question. On average, the participants started with their third question after 10.55 minutes. After the fourth question the participants were able to answer all questions faster than the average time taken by all participants to answer the respective questions, which corresponds to 18.47 minutes until effective usage of the tool.

#### 7.6.4.2 Correctness of Answers

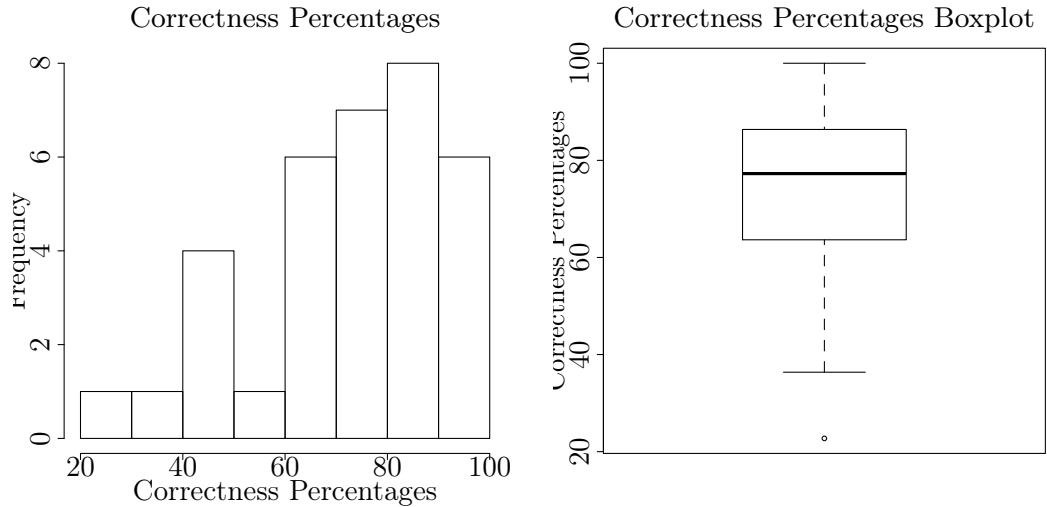
We also analyzed the participants' answers for correct results to compute correctness percentages over all their questions. Figure 7.13a shows the participants' percentages of correct answers from which we can see that most participants answered more than 60% of the question correctly. Figure 7.13b shows a box-and-whisker plot for the correctness percentages achieved, which shows that the highest score achieved was 100% with a median score of 78%.

We performed the same analysis as in Figure 7.12 using the correctness scores instead of the timing information. Figure 7.14b shows that more participants got their first question incorrect, regardless of which question they received first. However, as can also be seen in Figure 7.14c, after the first question there is no other indication that the sequence of the question affected the correctness of the participants answers.

#### 7.6.4.3 Results per Question

Figure 7.15 shows a box-and-whisker plot for each individual question showing the variation in times required per question. In the heading of each box and whisker diagram are indicated the average percentages that were achieved by all participants and whether the corresponding question required making use of a more complex tool functionality (F6 - the use of a sticky tag viewer, F7 - the use of a citation or reference clouds, F8 - the use of the table view) to answer the question.

We can see that no questions are outliers, in terms of time taken to answer the question, and that there is no identifiable pattern between the use of an advanced



(a) Histogram of Correctness Percentages Achieved (b) Boxplot of Correctness Percentages Achieved

Figure 7.13: Correctness Percentages Achieved

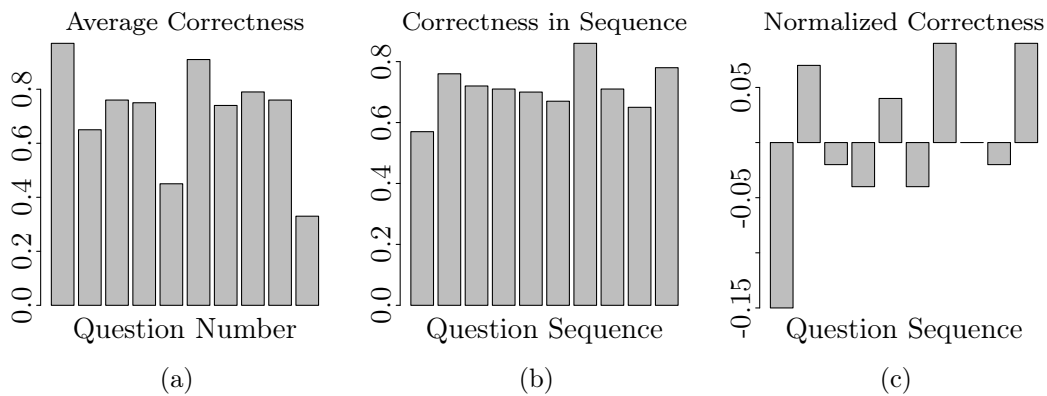


Figure 7.14: Average correctness for questions in (a) question order (b) the sequence that participants received them (c) the sequence that participants received them normalized by the average correctness that all participants obtained for the corresponding questions.

tool functionality and the participants' completion time. From the results of the individual questions presented in Figure 7.15, we see that Q5 and Q10 were the only questions in which the participants obtained an average correctness percentage below 50%. While question Q5 required the construction of a citation cloud, question Q10 only required straight-forward tool functionalities but the question itself was more complex and required the participants to suggest a relevant university for a particular author to collaborate with. Note that some questions (e.g., Q9 and Q10) have outliers in terms of the average completion time, which could be due to the random order in which participants received the questions.

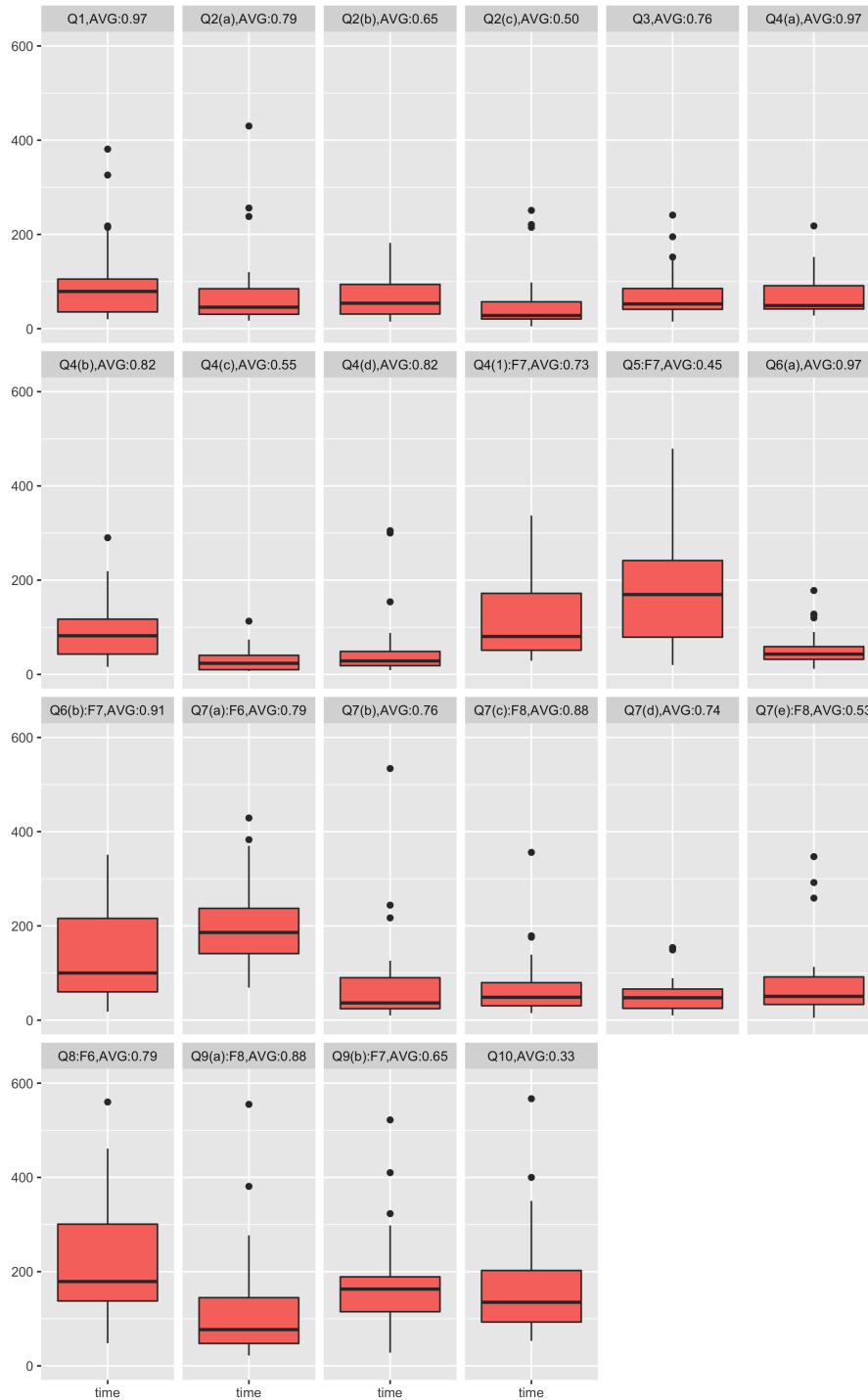


Figure 7.15: Boxplots per individual question. The question number is indicated in the heading for each boxplot along with the average correctness achieved for the question. The heading of each boxplot also indicates whether the question required functionalities sticky tag viewers (F6), citation or reference clouds (F7) or the table view (F8).



#### 7.6.4.4 Participant Feedback

After each participant completed the user study we asked them to fill out an anonymous feedback form about their experience using the tool. We asked participants to rate the different tool features on a Likert scale (1-5) according to how useful the participant found them and how easy the participant found the feature to use. In addition, the participants were asked to give feedback on any areas of the tool that could be improved and provide any additional comments about their experience using the tool.

Table 7.3 shows the mean and median scores for each tool feature. While participants found selecting tags and searching for tags both useful and easy to use, the feedback for filtering the tag cloud views to show only certain categories was indicated as useful but not as easy to use. Filtering the number of tags shown in a particular view was indicated as less useful but still easy to use. However, the sticky tag views were shown to be less useful and more difficult to use. The citation and reference clouds were indicated as the most difficult features to use but were rated as more useful than both the filtering of the tag clouds according to tag category and tag count. Participants also scored the tool with a mean of 4.49, when asked to indicate how likely they would use the tool in their own research.

Feature	Mean Usefulness	Median Usefulness	Mean Ease of Use	Median Ease of Use
(F1) Selecting/Deselecting Tags	4.27	5	4.64	5
(F2) Searching for Tags	4.86	5	4.46	5
(F3) Filtering Categories and creating category views	4.46	5	4.00	4
(F4) Filtering the number of tags	3.55	4	4.41	5
(F6) Sticky Tags	3.59	4	3.91	4
(F7) Citation and reference clouds	4.18	4	3.82	4

Table 7.3: Mean and median of participant scores for the usefulness and usability of tool features.

We used a process of open coding on the free-text feedback answers provided by participants to the questions “Are there any areas of the tools you feel could be improved” and “Please provide any comments on your experience of the tool”. We identified the major themes of the participants responses and noted that out of the 32 participants who were willing to provide anonymous feedback on the tool the themes identified were as follows.

47% of participants felt that some additional functionality would have improved their experience of the tool. For example one participant noted that they would like to “[be] able to create a new viewer from a selected tag. [They] could search for a tag, but [it] was unclear on how to make that tag a new viewer.”. 40% of participants indicated that they had difficulty understanding at least one feature of the tool. These included aspects such as whether the tag counts were derived from citation counts or publication counts and the effect of filtering the number of tags.

25% percent of participants indicated that at least one function of the tool did not behave as expected and 25% also indicated that they experienced a definite learning curve in their use of the tool. For example, one participant noted that “Once [they] got the hang of it, [they] found it very user friendly and can imagine it being very useful for researchers...”.

Participants labeled the tool as useful (38%), easy to use (19%), powerful (19%) and fun to use (16%). For example, participants noted that “[t]his could come in as a very useful tool in the future”. There were also some participants that reported feeling impartial to the tool (6%) and found it complicated (6%).

## 7.6.5 Analysis of Results

### 7.6.5.1 Effect of participant’s background on correctness results achieved

We performed one-way ANOVA tests to determine whether a participant’s background (years of research and academic rank) had any effect on the time required to answer all questions or the correctness percentage they achieved. We first performed a Shapiro-Wilk test on the participants timing information to confirm that the timing information was normally distributed and therefore appropriate for an ANOVA test.

When grouping the participants according to their years of research experience we found that the number of years of research experience did not make a statistically significant difference to the time it took the participants to answer the questions (p-value 0.937). When grouping participants into their academic ranks of Masters, PhD and Post-Doc we found that the academic rank also did not have a statistically significant effect on the time taken to answer questions (p-value 0.306).

We also tested whether the participants’ academic ranks and years of research experience made a statistically significant difference to their correctness percentages obtained. However, since the correctness percentages were not normally distributed (according to the Shapiro-Wilk test) we performed a Kruskal-Wallis test instead of an ANOVA test. We found that the correctness percentages were not statistically significantly different for any of the academic ranks of the participants as we could not

reject the null hypothesis (p-value 0.123). We also tested whether the participants' years of research affected their correctness percentages and again could not reject the null hypothesis (p-value 0.129), indicating no statistically significant difference between the correctness results of participants with different years of research.

#### **7.6.5.2 Effect of use of a specific tool function on correctness results achieved**

We also evaluated whether the required use of a specific more complex tool feature influences the correctness percentage achieved.

We tested this effect using a Pearson Chi-Square test since, for each question, we had categorical data stating whether or not a specific tool function was required. We found that for all questions requiring more complex tool functions F6 (sticky tags), F7 (citation and reference clouds) and F8 (use of the table view) we could not reject the null hypothesis that there is no association between the presence of the tool function and the correctness achieved. For F6, F7 and F8 the p-values were 0.27, 0.1 and 1.0 respectively. Therefore, we see that the involved tool function does not affect the correctness results.

#### **7.6.6 Threats to Validity**

The user study was conducted using a centralized lab server and so it is possible that some participants experienced slower loading times than others. However, since there were no time limits to specific questions in this study this is unlikely to have affected the user study responses.

Answers to questions in the user study were marked by us, which can introduce subjectiveness, especially since some questions were stated vaguely and multiple correct answers existed. For example, the interpretation of "prominent author" will inevitably vary between participants. However, the participant's answers were only marked as correct, if their understanding of "prominent author" was plausible scientifically.

The participants involved in the user study might not be representative of a real-world sample of academic researchers. However, all participants were pursuing post-graduate degrees or were post-doctoral researchers at the time of the study.

The results from the feedback of participants described in Section 7.6.4.4 might contain response biases.

## 7.7 Related Work

In this chapter we explore only work specifically related to browsing academic publication collections.

SurVis [40] allows users to construct an interactive visualization of paper collections that appear in literature survey papers. SurVis also provides a word cloud visualization for papers contained in a literature survey where different types of meta-information are presented in separate word clouds. Our publication browser also allows different categories to be separated into their own tag clouds as well as providing the option to have all tags in the same tag cloud, differentiated by color.

The ASE tool [67] allows users to familiarize themselves with a new research field. ASE operates on “citation text” which details citing papers that refer to the paper’s content compared to our approach of using text extracted from the papers’ titles and abstracts. ASE presents multiple inter-linked views (as can also be constructed in ConceptCloud) showing citation network visualizations as well as citation numbers for a specific group of papers. An in-cite summary which lists text surrounding a citation is also present in ASE to provide an overview of a paper. ASE operates on a smaller subset of data compared to the ACM DL dataset and therefore the goals of the analysis are somewhat different. The tool allows users to explore a subset of papers that have been retrieved through other means (search) as opposed to functioning as an interface for discovering new fields and papers in the broader library.

Dunne et al. [67] also performed a non-comparative user study for which they recruited participants that were researchers in computational linguistics, which is the same field as the papers in the dataset they used for their tool. Their study asked participants to “(a) Identify and make note of important authors and papers and (b) find an important paper and collect evidence to determine why it is important”. We posed a similar question in our study which also asks the participants to identify the important authors in a field (Q1). However, we have extended our question set to cover a wide variety of scientometric topics that are of interest in the literature. Alternative evaluation approaches have included deploying the evaluated tool and collecting usage data [64]. While we have not followed this approach, we have observed and taken note of each participant’s usage of our tool individually to identify whether they were able to use the functionalities successfully.

Medlar et al. [107] developed the PULP system for exploratory search of scientific literature. PULP presents information about topics covered in graphical format, where topics for each year are listed and lines between the different years are used to indicate how the topics over the different years are related. The user can select topics in the visualization to show keywords associated with the topic and once these

selections have been made a search interface is used to display academic papers related to the selections. The user then provides relevance feedback on the set of documents that has initially been provided and can then load more documents which will be based on which of the previous documents were marked as relevant. PULP also supports an exploratory search approach and allows users to select topics from a visualization as we do in our approach as well. However, we also allow users to select other fields of interest such as authors or affiliations. We also present the titles of the papers in a list format but present the full list to the user and allow them to continue refining the list by making additional tag selections.

Dörk et al. [64] use a “strolling” technique which allows users to explore publication data. Instead of refinements the PivotPaths tool uses pivots to change the information currently displayed in the browser, to allow users to not only explore a dataset by making refinements but also by navigating laterally in the dataset. Using our ConceptCloud browser, tags in the tag cloud can be de-selected in any order (not simply as an undo operation) which facilitates lateral navigation in the concept lattice, and therefore also in the underlying dataset.

Osborne et al. [112] do not make use of key-phrases but instead use the OWL ontology supplied by the Klink algorithm. Rexplore uses the ontology relation between papers to define the broaderGeneric or relatedEquivalent relationships between papers. Rexplore also provides a multiple interlinked view interface which describes the various facets in different views. The tool provides a topic analysis, which graphs how a topic has evolved over time as well as providing an author view which can indicate collaborative patterns.

Osborne et al. [112] performed a non-comparative user study in which they posed four tasks (including a warm-up task) to participants. One task that had to be completed was formulated as follows: “Find the top 5 authors with the highest number of publications in *Semantic Web* and rank them in terms of number of publications in *Artificial Intelligence*. For each of them find their most cited paper.” Their study is only based on the fields of “Semantic Web” and “Artificial Intelligence” and they do not use a cross-disciplinary library of publications, such as the one our study is based on. Therefore, our questions are formulated less restrictively and are closer to real-world questions that researchers would ask.

## 7.8 Conclusions

We have built an academic publication browser on top of the ConceptCloud browser framework. The publication browser supports users in exploring the publication data and is flexible enough to allow users to answer complex scientometric questions

which are difficult to answer using traditional list-based interfaces. The browser also presents aggregated citation and reference data to users, both on an individual paper basis and also for broader attributes, such as an author, conference or key-phrases which is novel.

The results of our user study show that participants were able to answer complex scientometric questions with an average correctness of 73%. The set up of our user study also enabled us to evaluate the learning curve that is associated with the ConceptCloud browser. We see that participants take on average three questions before they can use the browser efficiently. However, while participants were slower than average in the beginning of the study they only answered the first question less accurately than they did the rest of the question set.

We developed a new mechanism for handling references and citations in the concept lattice and made this functionality available in the publication browser. Participants gave the reference and citation clouds an average score of 4.18 for the usefulness of the clouds and 3.82 for the ease of use. Therefore, while participants recognized the usefulness of the reference and citation clouds the usability of this approach could still be improved in the interface. However, even though the usability of the citation and reference clouds was not rated highly by participants there is no statistically significant difference in the correctness between questions requiring the use of citation clouds and all other questions.

The publication browser application also demonstrates the scalability of our approach, which was able to handle over 1.8 million academic papers.

## Chapter 8

# Conclusions

### 8.1 Conclusions

Exploratory search fulfills a different purpose to traditional retrieval systems which require the user to provide a search term (or query) and often do not factor in the entire information seeking process which can include multiple search queries and lengthy browsing of the query results. In this dissertation we have designed and evaluated a generic framework to support exploratory search tasks on a variety of *semi-structured data archives* (i.e., data that comprises both structured and free text fields, where the most interesting information is often in the free text). Our framework supports exploratory search tasks which allow users to interact with the underlying data even when they have no previous knowledge of a dataset, or have not yet clearly defined their search task.

Our approach makes use of a novel combination of concept lattices and tag clouds in order to index and intuitively present semi-structured data collections that support interactive navigation. On selection or de-selection of tags in the tag cloud the current focus concept in the underlying concept lattice is updated and an updated tag cloud refined or broadened with the new selection is rendered. We have developed a broadening navigation approach in concept lattices which can be used in conjunction with the traditional refinement navigation techniques in order to support more flexible navigation. Our tag cloud interface presents aggregated information of large data collections directly to users so that they do not need to manually aggregate the information presented as they do when it is presented in traditional list-based interfaces. Our approach is also flexible enough to be applied to a variety of semi-structured data archives as the concept lattice provides a standard structure which can be populated with different datasets.

We have evaluated our framework by instantiating it using three different datasets: software development repository data, skills data extracted from GitHub profiles and

academic publication data from a large digital library collection. The data in each domain has different properties and therefore the instantiation of our framework in each domain has driven further development of the approach itself. Each instantiation of the framework makes novel contributions to its respective domain and the application of the framework in different domains also provides an evaluation for the flexibility of the generic framework itself.

Our software development archive browser allows users to explore the information contained in Git/SVN repositories as well as issue trackers in order to get an overview of the history of a software project. Users can explore the history of a software project in order to answer questions such as “what has changed in this project since I went on vacation?”. We evaluated our software development archive browser by applying it to two open-source projects and two industrial projects in the form of case studies where we were able to make relevant observations about the projects. We also performed a user study with the third year software engineering class at Stellenbosch University in order to evaluate whether new users were able to answer questions about unknown software projects using our browser. This study showed that participants were able to answer questions on the software history statistically significantly better using our ConceptCloud browser than when using a linear list view of commits as provided by the GitK tool.

Our CVExplorer skills browser presents aggregated skills information derived from collections of projects on GitHub. Our skills browser allows users to filter a large pool of developers to only those that possess the relevant skills. It extracts skills information directly from a developer’s commits to repositories on GitHub and provides developer skills that are more reliable (in the form of a portfolio) and fine-grained than those typically found on self-authored profiles, such as those available on LinkedIn. The GitHub interface itself does not allow easy identification of a developer’s skills as these are not presented in an aggregated form and so our browser provides information to users that has previously been difficult to access. We evaluated our skills browser by using it to recommend candidates for open positions at two different companies.

Our publication browser allows users to flexibly explore a set of publications and supports them in answering complex scientometric questions. We have developed a mechanism for handling object references using concept lattices which enables us to allow users to explore the aggregated references and citations of individual papers as well as broader attributes such as authors or conferences. We evaluated our publication browser by conducting a user study with post-graduate students from different departments at Stellenbosch University. Our user study showed that participants were able to use ConceptCloud to answer complex scientometric questions with an



average correctness score of 73%.

We have developed a flexible and scalable generic framework that can be used to generate concept lattices from a variety of different semi-structured data sources and make these available for exploratory search. We have also applied our framework to three different data sources that each have different properties in order to demonstrate its usefulness.

### Main Contributions

The main contribution of this PhD dissertation is the development of a framework that supports exploration tasks on a variety of semi-structured data sources and the application and evaluation of this framework in three different domains. We make novel contributions which include an approach for constructing an interactive tag cloud from data contained in a formal concept lattice, a broadening navigation approach in concept lattices and the development of the framework itself. Each instantiation of our framework also makes a contribution to its respective domain.

- Our software archive browser allows flexible exploration of software development archives without the need to specify a direct query. This approach has previously not been supported by other tools for mining software repositories.
- Our CVExplorer tool presents aggregated skills information for developers which is not easily accessible directly from their GitHub profiles, and can be used to aid candidate sourcing. The extraction and presentation of developer skills directly from their code contributions and the README files of projects they have contributed to is also novel.
- Our publication browser presents aggregated information for a large collection of publications which can be refined on various facets, such as authors, venues, countries, key-phrases and years. Our publication browser allows users to easily get an overview of the authors contributing to a topic and to access publications by selecting relevant key-phrases. Our browser also presents aggregated citation and reference data which is unavailable in traditional list-based interfaces.

## 8.2 Further Work Directions

There are possible extensions to our generic framework, and instantiated versions of the framework that have not been part of the research goals in this dissertation. Further extensions could make the browser even more intuitive for applications to other data sources not discussed in this dissertation.

### 8.2.1 Generic Framework

Our generic framework could be extended to provide additional visualizations as well as other layouts of tag cloud views. Additional steps in terms of context construction could also be provided in order to identify and exploit additional information in the data source.

#### Graph Visualizations

Our interface could be extended to include other visualizations, such as graphs, which could also be dynamically created for different categories of information. Graphs could also be created from particular sticky tag selections to graph, for example, the number of papers by a particular author in the past ten years. All tag cloud visualizations could also dynamically be converted into different visualizations, such as a tree map or bar-chart, by the user. Navigation could still be supported in the graphs as different aspects of the graphs could be selected to refine the views in the interface.

#### Tag Cloud Layouts

Different tag cloud layouts which are derived from the underlying concept lattice could be supported by the browser. Tags could be clustered according to the position of their introducing concepts in the underlying lattice to present more structure in the tag cloud view.

#### Extracting Additional Structure from the Data

Some data sources contain data that can be formatted in an ontology structure. For example, in wine review data, different wine varieties can be classed as red or white wines and different flavors could be represented hierarchically such as `raspberry` being a refinement of the `berry` flavor. If this ontology structure could be extracted from the original data source or additional domain-specific data sources then an ontology could be used to build context tables that allow even more intuitive navigation. For example, given the information that `raspberry` is a kind of `berry` we could apply the attribute `berry` to all wines that have been reviewed as being `raspberry` flavored.

#### Presenting Multiple Concept Lattices Simultaneously

Where different kinds of context tables (and therefore concept lattices) can be constructed for the same underlying data source (e.g., revision and file-based contexts from version control repository data), tag clouds derived from these different contexts could be presented simultaneously in the browser in different sections of the interfaces. Navigation could then also be linked across the different lattices so that if the user selects, for example, a file tag then both tag clouds derived from the revision and file-based contexts could be updated to show the selection in the different

lattices. In this way views of both the commit activity, derived from a revision-based context, and collaboration, derived from a file-based context, on different files could be presented and navigated at the same time.

### **Improving Usability Aspects**

Usability aspects that have been mentioned from feedback during the user studies could also be incorporated into the browser's interface. For example, the ease of creation of different tag clouds such as the sticky tag cloud views and the reference and citation clouds could be improved to make this functionality more intuitive for users.

### **Supporting User Customization of Contexts On-The-Fly**

Different user customizations on the underlying context table and, therefore concept lattice, could also be supported in the browser. For example, users could be allowed to merge two different tags in the browser on-the-fly, (such as two commit names which are both aliases for the same software developer). This kind of operation could then be used to merge the attributes in the underlying context table, so that the tag cloud view presents the newly merged attributes as one attribute.

## **8.2.2 Software Development Archive Browsing**

Our archive browser could be extended to combine more types of information such as information from continuous integration systems or messaging platforms. We have already shown that our approach can be used to successfully combine information from bug archives and version control repositories. For large software repositories that take a long time to index we could also extend the framework to allow incremental updates from software repositories so that only the additional commits in the software repository need to be processed when a version control repository is reloaded.

## **8.2.3 Skills Browsing**

For our CVExplorer tool we could use additional mechanisms to identify the location of a developer so that we could widen the candidate pool for a specific location as not all users indicate a location on their GitHub profiles. We could use the timezone information provided in the commit logs to verify a user's location (for which an exploratory study has already been discussed by Barahona et al. [84]) as well as using other profiles in conjunction with the GitHub profiles to identify the user's location. We could also aggregate skills from other platforms, such as Stack Overflow, as well to build a more complete picture of a candidate.

Our skills identification could also be extended to use the actual code changes that the developer makes and not just the changed file types. We could use the code changes to identify which libraries a developer is familiar with and then associate the developer to other wider skills such as “concurrency”. Therefore, our skills identification could be even more accurate and also provide a finer-grained skill set which can be used to more accurately filter a larger candidate pool. We could use the popularity of a project on GitHub and the other information on the quality and size of the code contributions to provide a weighting for different commits which might be of different qualities.

#### 8.2.4 Publication Browsing

Our publication browser could be extended with additional features that would allow users to export a list of the publications of interest, that they have discovered using the browser. The usability of the citation and reference clouds could also be further explored as participants of the user study indicated these features as the among those that were harder to use. Performance improvements for the construction and display of the citation and reference clouds could also be further explored so that citation clouds could load instantaneously.

The browser could also be applied to allow users to examine the publication data from particular events so that for example, conference attendees could use the browser to decide which paper presentations to attend.

#### 8.2.5 Additional Applications

Our framework could also be applied to a variety of other domains, to enable the flexible exploration of other semi-structured data collections. Our browser could be applied to different kinds of documents, such as news archives, to support exploratory search. Our browser could also be applied to product catalogs to support exploratory search tasks in domains such as on-line shopping.

### 8.3 Final Remarks

We have developed a generic framework, which has allowed us to make the information contained in version control repositories accessible to users, allowed browsing of developers’ skill sets as exhibited on GitHub and allowed flexible browsing of academic publications, as well as, their reference and citation data. Our framework uses a novel combination of tag clouds and formal concept lattices to support exploratory search tasks on semi-structured data archives, where the data is traditionally difficult to access and query.

# Appendices

## Appendix A

# ConSL Grammar

```
program ← definition+
        statement+
        layout*

definition ← define var_id as var_id {
            assign*
          }

layout ← group | layout_spec
assign ← var_id = string;
group ← group var_id+ as var_id;

layout ← layout var_id {
        assign+
      }

statement ← for loop | open
open ← open varid (string var_id)

for_loop ← for var_id in ['string'](where string)? {
          statements+
        }

var id ← letter (letter | number)*
letter ← (a|b|c|... |x|y|z)
number ← (0|1|... |8|9)
```

# List of References

- [1] Backbone. <https://github.com/jashkenas/backbone>.
- [2] Bitbucket. <http://bitbucket.com>.
- [3] Bus factor. <http://deviq.com/bus-factor/>.
- [4] Coderwall. <https://coderwall.com/>.
- [5] Eclipse jgit. <http://www.eclipse.org/jgit/>.
- [6] Egit github. <https://github.com/eclipse/egit-github/>.
- [7] Flickr. <https://www.flickr.com/>.
- [8] Github. <http://github.com>.
- [9] Github advanced search. <https://github.com/search/advanced>.
- [10] Github api. <https://developer.github.com/v3/>.
- [11] Github archive. <https://www.githubarchive.org/>.
- [12] Github press. <https://github.com/about/press>.
- [13] Gitk. <https://git-scm.com/docs/gitk>.
- [14] Google. <http://google.com>.
- [15] Google bigquery. <https://cloud.google.com/bigquery/>.
- [16] How to recruit a passive candidate. <http://careers.stackoverflow.com/resources/passive-candidate>.
- [17] Junit. <https://github.com/junit-team/junit4>.
- [18] Kibana. <https://www.elastic.co/products/kibana>.
- [19] Linkedin. <http://linkedin.com>.

- [20] LinkedIn press. <https://press.linkedin.com/about-linkedin>.
- [21] LinkedIn recruiter. <https://business.linkedin.com/talent-solutions/recruiter#>.
- [22] LinkedIn student sample profile. <https://www.linkedin.com/in/studentsample>.
- [23] Linux github repository. <https://github.com/torvalds/linux>.
- [24] Masterbranch. <https://www.masterbranch.com/>.
- [25] Open hub. <https://www.openhub.net/>.
- [26] Play framework. <https://www.playframework.com/>.
- [27] Retrofit. <https://github.com/square/retrofit>.
- [28] Rubygems. <https://github.com/rubygems/rubygems>.
- [29] Stack overflow candidate search. <http://business.stackoverflow.com/careers/us/platform/candidate-search>.
- [30] Stack overflow careers. <http://careers.stackoverflow.com/>.
- [31] Tag cloud example. [https://en.wikipedia.org/wiki/Tag\\_cloud#/media/File:State\\_of\\_the\\_union\\_word\\_clouds.png](https://en.wikipedia.org/wiki/Tag_cloud#/media/File:State_of_the_union_word_clouds.png).
- [32] Talentbin. <https://www.talentbin.com/>.
- [33] H. A. Abt. The future of single-authored papers. *Scientometrics*, 73(3):353–358, 2007.
- [34] ACM Digital Library. Acm digital library. <http://dl.acm.org/>, 2016.
- [35] I. F. Aguillo, J. Bar-Ilan, M. Levene, and J. L. Ortega. Comparing university rankings. *Scientometrics*, 85(1):243–256, 2010.
- [36] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Conference on Human Factors in Computing Systems*.
- [37] M. Alam, N. L. T. Nhu, and A. Napoli. Latviz: A new practical tool for performing interactive exploration over concept lattices. In *International Conference on Concept Lattices and Their Applications (CLA)*, 2016.



- [38] O. Alonso, P. T. Devanbu, and M. Gertz. Expertise identification and visualization from CVS. In *International Working Conference on Mining Software Repositories (MSR)*, pages 125–128, New York, NY, USA, 2008. ACM.
- [39] C. Anslow, J. Noble, S. Marshall, and E. Tempero. Visualizing the word structure of java class names. In *ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications (OOPSLA)*, pages 777–778. ACM, 2008.
- [40] F. Beck, S. Koch, and D. Weiskopf. Visual analysis and dissemination of scientific literature collections with survis. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):180–189, 2016.
- [41] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *International Conference on Software Engineering (ICSE)*, volume 1, pages 125–134. IEEE, 2010.
- [42] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is ?nearest neighbor? meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.
- [43] P. Buneman. Semistructured data. In *Symposium on Principles of Database Systems (PODS)*, pages 117–121, New York, NY, USA, 1997. ACM.
- [44] A. Capiluppi, A. Serebrenik, and L. Singer. Assessing technical candidates on the social web. *IEEE Software*, 30(1):45–51, Jan 2013.
- [45] J. Carbonnel, K. Bertet, M. Huchard, and C. Nebut. Fca for software product lines representation: Mixing configuration and feature relationships in a unique canonical representation. In *Concept Lattices and their Applications (CLA)*, pages 109–122. CEUR Workshop Proceedings, 2016.
- [46] L. D. Caro, K. S. Candan, and M. L. Sapino. Navigating within news collections using tag-flakes. *Journal of Visual Languages and Computing*, 22(2):120 – 139, 2011.
- [47] C. Carpineto and G. Romano. Order-theoretical ranking. *Journal of the American Society for Information Science*.
- [48] C. Carpineto and G. Romano. Ulysses: A lattice-based multiple interaction strategy retrieval interface. In *International Conference on Human-Computer Interaction*, pages 91–104. Springer, 1995.

- [49] C. Carpineto and G. Romano. Effective reformulation of boolean queries with concept lattices. In *Flexible Query Answering Systems*, pages 83–94. Springer, 1998.
- [50] C. Carpineto, G. Romano, and F. U. Bordoni. Exploiting the potential of concept lattices for information retrieval with credo. *Journal of Universal Computer Science (J. UCS)*, 10(8):985–1013, 2004.
- [51] X. Chen and A. Sarma. Supporting comparison of developer profiles across online communities. Technical report, Oregon State, 2016.
- [52] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey. Software history under the lens: a study on why and how developers examine it. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–10. IEEE, 2015.
- [53] V. Codocedo, I. Lykourantzou, and A. Napoli. A semantic approach to concept lattice-based information retrieval. *Annals of Mathematics and Artificial Intelligence*, 72(1-2):169–195, 2014.
- [54] R. Cole, P. Eklund, and G. Stumme. Document retrieval for e-mail search and discovery using formal concept analysis. *Applied artificial intelligence*, 17(3):257–280, 2003.
- [55] Connor, James. Scholar Updates: Making New Connections. <http://googlescholar.blogspot.co.za/2012/08/scholar-updates-making-new-connections.html>, 2012.
- [56] R. Cottrell, B. Goyette, R. Holmes, R. Walker, and J. Denzinger. Compare and contrast: Visual exploration of source code examples. In *International Workshop on Visualizing Software for Understanding and Analysis*, pages 29–32, Sept 2009.
- [57] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Transactions Software Engineering (TSE)*, 31(6):446–465, 2005.
- [58] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Conference on Computer-Supported Cooperative Work (CSCW)*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.
- [59] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2. ed.)*. Cambridge University Press, 2002.

- [60] DBPedia. Dbpedia. <http://wiki.dbpedia.org/>, 2016. Online; Accessed 24-Oct-2016.
- [61] D. J. de Solla Price. Networks of Scientific Papers. *Science*, 149(3683):510–515, 1965.
- [62] S. DeKay. Are business-oriented social networking web sites useful resources for locating passive jobseekers? results of a recent study. *Business Communication Quarterly*, 72(1):101–105, 2009.
- [63] T. DeMarco and T. Lister. *Peopeware: Productive Projects and Teams*. Pearson Education, 2013.
- [64] M. Dörk, N. H. Riche, G. Ramos, and S. Dumais. Pivotpaths: Strolling through faceted information spaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2709–2718, Dec 2012.
- [65] D. Doubrovkine. Github is your new resume. <http://code.dblock.org/2011/07/14/github-is-your-new-resume.html>.
- [66] M. Dunaiskii, G. J. Greene, and B. Fischer. Exploratory search of academic publication and citation data using interactive tag cloud visualizations. 2016.
- [67] C. Dunne, B. Shneiderman, R. Gove, J. Klavans, and B. Dorr. Rapid understanding of scientific paper collections: Integrating statistics, text analytics, and visualization. *Journal of the American Society for Information Science and Technology (JASIST)*, 63(12):2351–2369, 2012.
- [68] C. Eccles. The use of university rankings in the united kingdom. *Higher Education in Europe*, 27(4):423–432, 2002.
- [69] P. Eklund, J. Ducrou, and P. Brawn. Concept lattices for information visualization: Can novices read line-diagrams? In *International Conference on Formal Concept Analysis (ICFCA)*, pages 57–73. Springer, 2004.
- [70] J. Emerson, N. Churcher, and C. Deaker. From toy to tool: Extending tag clouds for software and information visualisation. In *Australian Software Engineering Conference*, pages 155–164, 2013.
- [71] M. Esterhuizen. ConSL: Concept cloud scripting language, 2015.
- [72] M. Faid, R. Missaoui, and R. Godin. Knowledge discovery in complex objects. *Computational Intelligence*, 15(1):28–49, 1999.

- [73] S. Ferré. Camelis: a logical information system to organise and browse a collection of documents. *International Journal of General Systems*, 38(4):379–403, 2009.
- [74] B. Fischer. Specification-based browsing of software component libraries. *Automated Software Engineering (ASE)*, 7(2):179–200, 2000.
- [75] T. Fritz and G. C. Murphy. Using information fragments to answer the questions developers ask. In *International Conference on Software Engineering (ICSE)*, pages 175–184. ACM, 2010.
- [76] B. Ganter and R. Wille. *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*, chapter Conceptual Scaling, pages 139–167. Springer US, New York, NY, 1989.
- [77] B. Ganter and R. Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.
- [78] E. Garfield. Is Citation Analysis A Legitimate Evaluation Tool? *Scientometrics*, 1(4):359–375, 1979.
- [79] T. Girba, S. Ducasse, A. Kuhn, R. Marinescu, and R. Daniel. Using concept analysis to detect co-change patterns. In *International Workshop on Principles of Software Evolution (IWPSSE)*, pages 83–89, New York, NY, USA, 2007. ACM.
- [80] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *International Workshop on Principles of Software Evolution*, pages 113–122, Sept 2005.
- [81] R. Godin, C. Pichet, and J. Gecsei. Design of a browsing interface for information retrieval. *Special Interest Group on Information Retrieval Forum*, 23(SI):32–39, May 1989.
- [82] R. Godin, E. Saunders, and J. Gecsei. Lattice model of browsable data spaces. *Information Sciences*, 40(2):89–116, 1986.
- [83] M. Goeminne and T. Mens. A comparison of identity merge algorithms for software repositories. *Science of Computer Programming*, 78(8):971 – 986, 2013.
- [84] J. M. Gonzalez Barahona, G. Robles, and D. Izquierdo-Cortazar. Determining the geographical distribution of a community by means of a time-zone analysis. 2016.

- [85] D. N. Götzmann. Colibri/java. <http://code.google.com/p/colibri-java/>, 2007.
- [86] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.
- [87] J. Gwizdka and P. Bakelaar. Tag trails: navigation with context and history. In *Extended Abstracts on Human Factors in Computing Systems (CHI)*, pages 4579–4584. ACM, 2009.
- [88] M. J. Halvey and M. T. Keane. An assessment of tag presentation techniques. In *International Conference on World Wide Web (WWW)*, pages 1313–1314, New York, NY, USA, 2007. ACM.
- [89] C. Hauff and G. Gousios. Matching github developer profiles to job advertisements. In *International Conference on Mining Software Repositories (MSR)*, 2015.
- [90] A. Hindle and D. M. Germán. Scql: a formal model and a query language for source control repositories. *SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005. Proc. MSR’05.
- [91] J. E. Hirsch. An index to quantify an individual’s scientific research output. *Proceedings of the National Academy of Sciences*, 102(46):16569–16572, aug 2005.
- [92] S. E. Hoey. New research features on Mendeley.com! <https://blog.mendeley.com/2015/11/03/new-research-features-on-mendeley-com/>, 2015.
- [93] M. Huchard, C. Roume, and P. Valtchev. When concepts point at other concepts: the case of uml diagram reconstruction. In *Proceedings of the 2nd Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD)*, pages 32–43, 2002.
- [94] A. W. Kosner. Software engineers are in demand, and github is how you find them. <http://www.forbes.com/sites/anthonykosner/2012/10/20/software-engineers-are-in-demand-and-github-is-how-you-find-them/>.
- [95] C. C. Kuhlthau. Inside the search process: Information seeking from the user’s perspective. *Journal of the American society for information science (JASIST)*, 42(5):361, 1991.

- [96] S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):189–216, 2002.
- [97] C. Lindig. Concept-based component retrieval. In *Working notes of the IJCAI-95 workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs*, pages 21–25, 1995.
- [98] C. Lindig. *Algorithmen zur Begriffsanalyse und ihre Anwendung bei Softwarebibliotheken*. PhD thesis, TU Braunschweig, 1999.
- [99] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 19th international conference on Software engineering*, pages 349–359. ACM, 1997.
- [100] J. Loeliger and M. McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.
- [101] S. Lohmann, J. Ziegler, and L. Tetzlaff. Comparison of tag cloud layouts: Task-related performance and visual exploration. In *International Conference on Human-Computer Interaction (INTERACT)*, pages 392–404, 2009.
- [102] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. An Introduction to Information Retrieval. Cambridge University Press, 2008.
- [103] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.
- [104] J. Marlow and L. Dabbish. Activity traces and signals in software developer recruitment and hiring. In *Conference on Computer-Supported Cooperative Work (CSCW)*, pages 145–156. ACM, 2013.
- [105] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Conference on Computer-Supported Cooperative Work (CSCW)*, pages 117–128. ACM, 2013.
- [106] P. McCuller. *How to Recruit and Hire Great Software Engineers: Building a Crack Development Team*. Apress, Berkely, CA, USA, 1st edition, 2012.
- [107] A. Medlar, K. Ilves, P. Wang, W. Buntine, and D. Glowacka. Pulp: A system for exploratory search of scientific literature. In *Proceedings of the 39th*

- International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1133–1136, New York, NY, USA, 2016. ACM.
- [108] C. S. Mesnage and M. J. Carman. Tag navigation. In *International Workshop on Social Software Engineering and Applications (SoSEA)*, pages 29–32. ACM, 2009.
- [109] D. R. Millen and J. Feinberg. Using social tagging to improve social navigation. In *Workshop on the Social Navigation and Community based Adaptation Technologies*. Citeseer, 2006.
- [110] G. C. Murphy and D. Notkin. Lightweight lexical source model extraction. *ACM Transactions on Software Engineering Methodology*, 5(3):262–292, 1996.
- [111] R. Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2), 2009.
- [112] F. Osborne, E. Motta, and P. Mulholland. Exploring scholarly data with rexplore. In *The Semantic Web-ISWC 2013*, pages 460–477. Springer, 2013.
- [113] E. Pariser. *The filter bubble: What the Internet is hiding from you*. Penguin UK, 2011.
- [114] P. D. B. Parolo, R. K. Pan, R. Ghosh, B. A. Huberman, K. Kaski, and S. Fortunato. Attention decay in science. *Journal of Informetrics*, 9(4):734–745, 2015.
- [115] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version Control with Subversion - The Standard in Open Source Version Control*. O'Reilly, 2008.
- [116] M. F. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.
- [117] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *International Conference on Program Comprehension (ICPC)*, pages 37–48, 2007.
- [118] R. Prieto-Diaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):88–97, 1991.
- [119] R. Prieto-Diaz and P. Freeman. Classifying software for reusability. *IEEE software*, 4(1):6, 1987.

- [120] U. Priss. Lattice-based information retrieval. *Knowledge Organization*, 27(3):132–142, 2000.
- [121] U. Priss. Formal concept analysis in information science. *Annual Review of Information Science and Technology (Arist)*, 40(1):521–543, 2006.
- [122] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chianti: a tool for change impact analysis of java programs. In *ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications (OOP-SLA)*, pages 432–448, 2004.
- [123] D. Richards and P. Compton. Combining formal concept analysis and ripple down rules to support the reuse of knowledge. *representations*, 2:36, 1997.
- [124] M. Ringel, E. Cutrell, S. Dumais, and E. Horvitz. Milestones in time: The value of landmarks in retrieving information from personal stores. In *International Conference on Human-Computer Interaction*, volume 2003, pages 184–191, 2003.
- [125] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. *SIGSOFT Software Engineering Notes*, 30(4):1–5, May 2005.
- [126] M. Rosvall and C. T. Bergstrom. Mapping change in large networks. *PloS one*, 5(1):e8694, jan 2010.
- [127] D. Rusk and Y. Coady. Location-based analysis of developers and technologies on github. In *International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 681–685. IEEE, 2014.
- [128] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.
- [129] I. Schmitt and G. Saake. Merging inheritance hierarchies for database integration. In *Cooperative Information Systems, 1998. Proceedings. 3rd IFCIS International Conference on*, pages 322–331. IEEE, 1998.
- [130] J. Schrammel, M. Leitner, and M. Tscheligi. Semantically structured tag clouds: An empirical evaluation of clustered presentation approaches. In *Conference on Human Factors in Computing Systems (CHI)*, pages 2037–2040, New York, NY, USA, 2009. ACM.
- [131] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.



- [132] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 23–34, New York, NY, USA, 2006. ACM.
- [133] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Conference on Computer-Supported Cooperative Work (CSCW)*, pages 103–116. ACM, 2013.
- [134] V. Sinha and D. R. Karger. Magnet: Supporting navigation in semistructured data environments. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD '05*, pages 97–106, New York, NY, USA, 2005. ACM.
- [135] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *International Workshop on Mining Software Repositories (MSR)*, pages 1–5. ACM, 2005.
- [136] G. Snelling. Reengineering of configurations based on mathematical concept analysis. *ACM Transactions on Software Engineering Methodology*, 5(2):146–189, Apr. 1996.
- [137] G. Snelling and F. Tip. Reengineering class hierarchies using concept analysis. *SIGSOFT Software Engineering Notes*, 23(6):99–110, Nov. 1998.
- [138] I. Sommerville. *Software Engineering*. Pearson Education, 2016.
- [139] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- [140] S. Thummalapenta and T. Xie. Spotweb: Detecting framework hotspots and coldspots via mining open source code on the web. In *International Conference on Automated Software Engineering (ASE)*, pages 327–336, Sept 2008.
- [141] T. Tilley, R. Cole, P. Becker, and P. Eklund. A survey of formal concept analysis support for software engineering activities. In *Formal concept analysis*, pages 250–271. Springer, 2005.
- [142] J. W. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, pages 99–114, 1949.
- [143] C. Van Rijsbergen. *Information retrieval*. Butterworths, 1979.
- [144] J. Vesperman. *Essential CVS*. O'Reilly Media, Inc., 2006.

- [145] J. Vicknair, D. Elkersh, K. Yancey, and M. C. Budden. The use of social networking websites as a recruiting tool for employers. *American Journal of Business Education (AJBE)*, 3(11), 2010.
- [146] M. L. Wallace, V. Larivière, and Y. Gingras. A Small World of Citations: The Influence of Collaboration Networks on Citation Practices. *PLoS ONE*, 7(3):e33339, 2012.
- [147] B. Weiss. Github is your resume now. <http://anti-pattern.com/github-is-your-resume-now>.
- [148] P. Weissgerber, M. Pohl, and M. Burch. Visual data mining in software archives to detect how developers work together. In *International Workshop on Mining Software Repositories (MSR)*, pages 9–, Washington, DC, USA, 2007. IEEE Computer Society.
- [149] J. D. West, M. C. Jensen, R. J. Dandrea, G. J. Gordon, and C. T. Bergstrom. Author-level eigenfactor metrics: Evaluating the influence of authors, institutions, and countries within the social science research network community. *Journal of the American Society for Information Science and Technology (JASIST)*, 64(4):787–801, 2013.
- [150] R. W. White, B. Kules, S. M. Drucker, and m. schraefel. Introduction. *Communications of the ACM*, 49(4):36–39, Apr. 2006.
- [151] R. W. White and R. A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.
- [152] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Reidel, 1982.
- [153] R. Wille. Why can concept lattices support knowledge discovery in databases? *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):81–92, 2002.
- [154] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Conference on Human Factors in Computing Systems (CHI)*, pages 401–408, New York, NY, USA, 2003. ACM.
- [155] A. Zaidman, B. Van Rompaey, S. Demeyer, and A. Van Deursen. Mining software repositories to study co-evolution of production & test code. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 220–229. IEEE, 2008.

- [156] M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998.