

# **Qualitative Repository Analysis with RepoGrams**

by

Daniel Rozenberg

B.Sc., The Open University of Israel, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL  
STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

August 2015

© Daniel Rozenberg, 2015

# Abstract

The availability of open source software projects has created an enormous opportunity for empirical evaluations in software engineering research. However, this availability requires that researchers judiciously select an appropriate set of evaluation targets and properly document this rationale. This selection process is often critical as it can be used to argue for the generalizability of the evaluated tool or method.

To understand the selection criteria that researchers use in their work we systematically read 55 research papers appearing in six major software engineering conferences. Using a grounded theory approach we iteratively developed a codebook and coded these papers along five different dimensions, all of which relate to how the authors select evaluation targets in their work. Our results indicate that most authors relied on qualitative and subjective features to select their evaluation targets.

Building on these results we developed a tool called RepoGrams, which supports researchers in comparing and contrasting source code repositories of multiple software projects and helps them in selecting appropriate evaluation targets for their studies. We describe RepoGrams's design and implementation, and evaluate it in two user studies with 74 undergraduate students and 14 software engineering researchers who used RepoGrams to understand, compare, and contrast various metrics on source code repositories. For example, a researcher interested in evaluating a tool might want to show that it is useful for both software projects that are written using a single programming language, as well as ones that are written using dozens of programming languages. RepoGrams allows the researcher to find a set of software projects that are diverse with respect to this metric.

We also evaluate the amount of effort required by researchers to extend RepoGrams for their own research projects in a case study with 2 researchers.

We find that RepoGrams helps software engineering researchers understand and compare characteristics of a project's source repository and that RepoGrams can be used by non-expert users to investigate project histories. The tool is designed primarily for software engineering researchers who are interested in analyzing and comparing source code repositories across multiple dimensions.

# Preface

The work presented in this thesis was conducted in the Software Practices Lab under supervision of Prof. Ivan Beschastnikh.

The user studies described in this thesis were approved by the UBC Behavioural Research Ethics Board under the certificate H14-02474.

# Table of Contents

<b>Abstract</b> . . . . .	<b>ii</b>
<b>Preface</b> . . . . .	<b>iv</b>
<b>Table of Contents</b> . . . . .	<b>v</b>
<b>List of Tables</b> . . . . .	<b>ix</b>
<b>List of Figures</b> . . . . .	<b>xi</b>
<b>Glossary</b> . . . . .	<b>xiii</b>
<b>Acknowledgments</b> . . . . .	<b>xiv</b>
<b>Dedication</b> . . . . .	<b>xv</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 Research questions . . . . .	5
1.2 Contributions . . . . .	6
<b>2 Related work</b> . . . . .	<b>8</b>
2.1 Selection of evaluation targets . . . . .	8
2.2 Literature surveys . . . . .	9
2.3 Software visualizations . . . . .	10
<b>3 Project selection approaches in Software Engineering (SE) literature</b>	<b>12</b>
3.1 Protocol . . . . .	12

3.2	Codebook . . . . .	13
3.3	Results . . . . .	17
<b>4</b>	<b>RepoGrams’s design and implementation . . . . .</b>	<b>21</b>
4.1	Design . . . . .	21
4.1.1	Visual abstractions . . . . .	23
4.1.2	Mapping values into colors with buckets . . . . .	24
4.1.3	Supported interactions . . . . .	27
4.2	Implementation details . . . . .	28
4.3	Implemented metrics . . . . .	29
<b>5</b>	<b>Evaluation . . . . .</b>	<b>34</b>
5.1	User study with undergraduate students . . . . .	35
5.1.1	Methodology . . . . .	35
5.1.2	Results . . . . .	36
5.1.3	Summary . . . . .	38
5.2	User study with SE researchers . . . . .	38
5.2.1	Methodology . . . . .	39
5.2.2	Results . . . . .	40
5.2.3	Semi-structured interview . . . . .	44
5.2.4	Summary . . . . .	47
5.3	Estimation of effort involved in adding new metrics . . . . .	47
5.3.1	Summary . . . . .	48
<b>6</b>	<b>Future work . . . . .</b>	<b>49</b>
6.1	Additional features . . . . .	49
6.2	Expanded audience . . . . .	50
6.3	Further evaluations . . . . .	51
<b>7</b>	<b>Conclusion . . . . .</b>	<b>52</b>
	<b>Bibliography . . . . .</b>	<b>53</b>

<b>A</b>	<b>Literature survey</b>	<b>63</b>
A.1	Full protocol	63
A.1.1	Scope	63
A.1.2	Overview	63
A.1.3	Procedure	64
A.2	Categories	65
A.2.1	Notes	67
A.3	Raw results	67
<b>B</b>	<b>Undergraduate students study</b>	<b>71</b>
B.1	Slides from the in-class demonstration	71
B.2	Protocol and questionnaire	77
B.2.1	Overview	77
B.2.2	Questionnaire	77
B.2.3	Demographics	78
B.2.4	Warmup questions	79
B.2.5	Metric comprehension questions	82
B.2.6	Questions about comparisons across projects	84
B.2.7	Exploratory question	86
B.2.8	Open comments	87
B.2.9	Filtering results	87
B.3	Raw results	87
<b>C</b>	<b>Software engineering researchers study</b>	<b>98</b>
C.1	Protocol and questionnaire	98
C.1.1	Procedure overview	98
C.1.2	Study protocol	98
C.1.3	Questionnaire	99
C.1.4	Filtering results	106
C.2	Raw results	106
<b>D</b>	<b>Case study</b>	<b>108</b>
D.1	Results	108
D.1.1	Overview	108

D.1.2	Raw results . . . . .	109
<b>E</b>	<b>License and availability . . . . .</b>	<b>110</b>



# List of Tables

Table 3.1	SE conferences that we reviewed in our literature survey. . . .	13
Table 3.2	Selection criteria codes and frequencies from our literature survey.	15
Table 3.3	Project visibility codes and frequencies from our literature survey.	18
Table 4.1	Alphabetical list of all metrics included in the current imple- mentation of RepoGrams. . . . .	30
Table 5.1	Main questions from the advanced user study. . . . .	40
Table A.1	Results on the initial set of 59 papers used to seed the codebook.	68
Table A.2	Results and analysis of the survey of 55 paper. . . . .	70
Table B.1	Raw results from the demographics section in the user study with undergraduate students. . . . .	88
Table B.2	Raw results from the warmup section (questions 1–4) in the user study with undergraduate students. . . . .	89
Table B.3	Raw results from the metrics comprehension section (questions 5–7) in the user study with undergraduate students. . . . .	90
Table B.4	Raw results from the metrics comprehension section (questions 8–10) in the user study with undergraduate students. . . . .	91
Table B.5	Raw results from the project comparison section (questions 11– 13) in the user study with undergraduate students. . . . .	93
Table B.6	Raw results from the exploratory question in the user study with undergraduate students. . . . .	95

Table C.1	Raw results from the user study with SE researchers. . . . .	107
Table D.1	Raw results from the case study to estimate the effort involved in the implementation of new metrics. . . . .	109

# List of Figures

Figure 1.1	The <i>repository footprint</i> visual abstraction . . . . .	3
Figure 1.2	Repository footprints for a section of repository histories of <code>sqlitebrowser</code> and <code>postr</code> projects . . . . .	4
Figure 3.1	Frequency of the number of evaluation targets by number of papers . . . . .	19
Figure 4.1	RepoGrams interface: (1) input field to add new projects, (2) button to select the metric(s), (3) a <i>repository footprint</i> corresponding to a specific project/metric combination. The color of a <i>commit block</i> represents the value of the metric on that commit, (4) the legend for commit values in the selected metric(s), (5) zoom control, (6) button to switch block length representation and normalization mode, (7) buttons to remove or change the order of repository footprints, (8) way of switching between grouping by metric and grouping by project (see Figure 4.4), (9) Tooltip displaying the exact metric value and the commit message (truncated), (10) metric name and description	22
Figure 4.2	All six combinations of block length and normalization modes	24
Figure 4.3	Examples of legends generated from buckets for the <i>Languages in the Commit</i> , <i>Files Modified</i> , and <i>Number of Branches</i> metrics.	25

Figure 4.4	RepoGrams supports two ways of grouping repository footprints: (a) the metric-grouped view facilitates comparison of different projects along the same metric, and (b) the project-grouped view facilitates comparison of the same project along different metrics. . . . .	27
Figure 5.1	RepoGrams showing the repository footprints as it was during the user study with undergraduate students, <b>question 5</b> . . . . .	37
Figure 5.2	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 4</b> . . . . .	41
Figure 5.3	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 5</b> . . . . .	41
Figure 5.4	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 6</b> . . . . .	42
Figure 5.5	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 7</b> . . . . .	43
Figure 5.6	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 8</b> . . . . .	43
Figure 5.7	RepoGrams showing the repository footprints as it was during the user study with SE researchers, <b>question 9</b> . . . . .	44

# Glossary

**SE** Software Engineering

**VCS** Version Control System

**CVS** Concurrent Versions System

**LoC** Lines of Code

**TA** Teaching Assistant

**ACM** Association for Computing Machinery

**ICSE** International Conference on Software Engineering

**MSR** Working Conference on Mining Software Repositories

**FSE** International Symposium on the Foundations of Software Engineering

**ASE** International Conference on Automated Software Engineering

**ESEM** International Symposium on Empirical Software Engineering and Measurement

# Acknowledgments

First and foremost I would like to thank my supervisor, Dr. Ivan Beschastnikh, for constantly pushing me forward. I am grateful to Dr. Gail C. Murphy and Dr. Marc Palyart who have both played a part as important as Ivan's or mine in this work.

I would also like to thank our colleagues from Saarland University: Fabian Kosmale, Valerie Poser, Heiko Becker, Maike Maas, Sebastian Becking, and Marc Jose for developing the initial versions of RepoGrams.

In addition, I convey my sincerest thanks to the 112 people who participated in and piloted our two user studies.

Finally, no acknowledgment section would be complete without thanking my labmates in both the SPL (Software Practices Labs) and the NSS (Networks, Systems, and Security) lab where I spent most of my time due to the presence of windows which the SPL lacked.

This work was supported by NSERC and by the UBC Computer Science department.

# Dedication

SoSwI'vaD vavwI'vaD je...  
...tuqmaj quvmoHjaj QeD ghItlhvam.

To my parents...  
... who support and encourage me in more ways than one.

# Chapter 1

## Introduction

Software Engineering (SE) researchers are increasingly using open source project information stored in centralized hosting sites, such as GitHub and Bitbucket, to help evaluate their ideas. Researchers who developed a tool or method as part of their research can evaluate them on artifacts from software projects in these centralized repositories, e.g., source code, execution logs, social meta-data. GitHub alone hosts tens of millions of projects, has about 9 million registered users and is one of the top 100 most popular websites in the world [9, 42]. Although there have been recent studies that take advantage of this enormous availability by evaluating artifacts from hundreds or thousands of projects from such repositories in their evaluations [17, 52], most studies rely on just a handful of evaluation targets that were picked manually by the researchers. We conducted a literature survey of 114 papers from major SE conferences, 65 papers of which evaluated their research with 8 or fewer evaluation targets, in part because of the detailed analysis needed for the questions being asked. For these studies the availability of projects has a flip side: although it became easier to access artifacts of software projects for use in evaluations, researchers now need a strategy for selecting the project or projects whose artifacts they will use in the evaluation of their tool or method.

To understand the existing project selection strategies we reviewed in detail 55 recently published SE papers and found that most of them relied on qualitative features of the evaluation targets that were selected. For example, consider two studies published at the Foundations of Software Engineering (FSE) 2012 confer-



ence in which the authors selected projects for their evaluation in ad hoc ways. The first study focused on detecting insecure component usage [37]; the authors describe their selection strategy as follows: “*we have analyzed [six] applications using widely-used components (such as the IE browser components and the Flash Player)*”. The paper does not clarify elsewhere why the authors selected these six specific applications. By failing to report the reasons that these applications were selected over any others, we the readers can only make educated guesses against which metrics of software projects this tool’s usefulness generalizes. e.g., whether the tool is useful when applied to software projects with a diverse number of programming languages, set of build tools, varying team sizes.

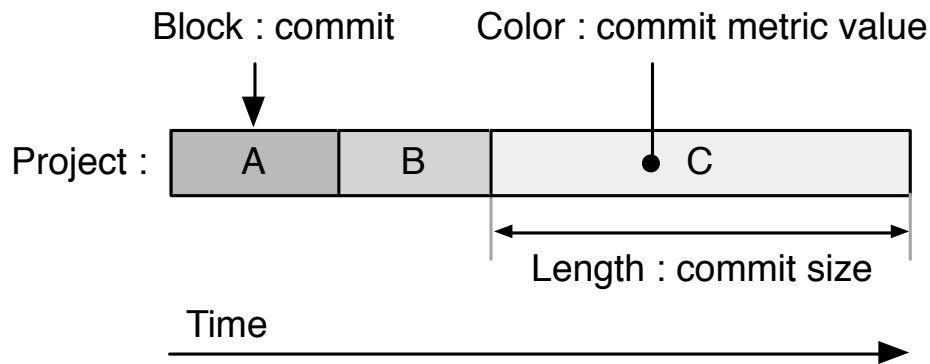
In another study from FSE 2012, the authors developed new approaches to summarize bug reports [40]. For their selection strategy, the authors relied on prior work and access to an internal IBM product: “*We conducted our experiments on two subjects. The first subject . . . was obtained from [34]. . . . The second subject was obtained from IBM DB2 team.*” Here, the rationale for selecting the subject DB2 is unclear. Moreover, it is not clear if the subject DB2 has added value over the subjects from prior work (e.g., whether or not it is substantially different in terms of bug reporting practices).

In our literature survey we found that SE researchers rarely state explicitly whether they employed a strategy to select evaluation targets, with notable exception in papers that evaluate the same targets as related works. It is therefore likely that researchers find evaluation targets in an haphazard manner and cannot report on their selection process. Based on these findings we believe that SE researchers could benefit from a tool designed to select appropriate software projects for their evaluations. Prior work has proposed new metric-based approaches to selecting diverse projects for research evaluation [45]. We add a different dimension to this perspective by proposing a tool to help support the existing *qualitative* selection practices in the SE community.

We designed, implemented, and evaluated a new tool, called RepoGrams. RepoGrams supports researchers in comparing and contrasting source code repositories of multiple software projects and helps them in selecting appropriate evaluation targets for their studies. Our evaluations, comprising of a literature survey of SE

papers, two user studies, and one case study, demonstrate that RepoGrams is useful to SE researchers who are interested in assessing and comparing the development activity of multiple software projects.

**Figure 1.1:** The *repository footprint* visual abstraction



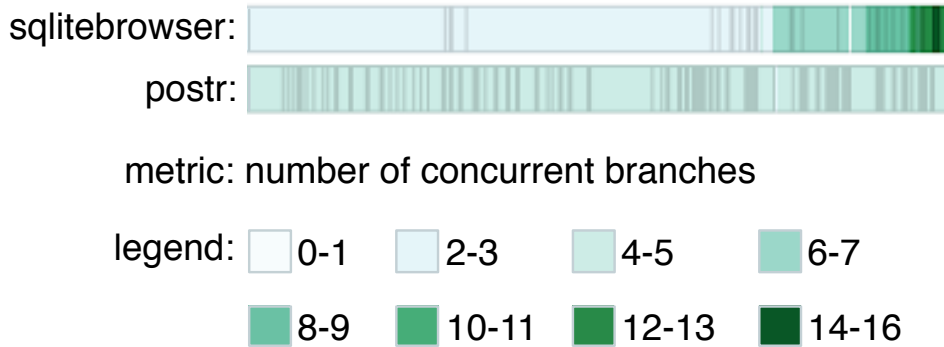
A key feature of RepoGrams is its ability to compactly present information about the history of the source code of multiple software projects in a way that is (1) relevant to a researcher's task, and (2) simplifies comparison. RepoGrams accomplishes this by computing user-selected (or user-implemented) metrics on software repositories. These metrics use artifacts in the Git repository (e.g., number of Lines of Code (LoC) changed in a commit, the commit author, the branch in which the commit occurred) to compute either scalar or discrete values. RepoGrams then visualizes the metric values using the *repository footprint* visual abstraction (Figure 1.1). A repository footprint (*footprint* for short) represents all the individual commits from a software project's source code repository as a sequence of blocks, one block per commit.

The blocks are laid out left-to-right in temporal order. In Figure 1.1 the commit represented by block A was committed to the repository before the commit of block B, and both were committed before commit of block C. Other than this strict ordering of the commits, the footprint does not reveal any more information regarding time. i.e., we cannot know how much time passed between the commits represented by blocks A and B based on their position.

A block's color and length represent the corresponding commit's value in the selected metric, and a user-selected mode respectively. One example of a block length mode is a linear correspondence between the commit's size in terms of LoC changed and each block's length.

Visual information is processed differently in the human mind. The visual system provides a high-bandwidth channel that processes information in parallel, compared to verbal information that is processed serially [43]. This allows users of the tool to comprehend the presented information faster than they would if the information was represented verbally or numerically.

**Figure 1.2:** Repository footprints for a section of repository histories of `sqlitebrowser` and `postr` projects



For example, Figure 1.2 shows the repository footprints of two projects: a `sqlitebrowser` [7] footprint (top) and a `postr` [5] footprint (bottom). In this example the metric is *Number of Branches*, a metric that counts the number of active concurrent branches at the time a commit was introduced. From this figure we can make two observations: (1) in contrast to `postr`, `sqlitebrowser` progressively uses more concurrent branches over time. We can see this by observing the footprints left to right: the footprint for `sqlitebrowser` becomes darker and it eventually has as many as 14–16 concurrent branches. In contrast, the footprint for `postr` does not change its color and remains in the range of 4–5 concurrent branches. (2) `sqlitebrowser`'s footprint contains commits that are significantly larger than those in `postr`. We can see this by comparing the length of the commit blocks between the two footprints.

A researcher might want, for example, to study why some development teams change the way they use branches over time. This researcher can use RepoGrams to identify different patterns of branch usage and select projects with repository footprints that exhibit irregular patterns to perform an evaluation on. If most of the projects in this researcher’s scope were to exhibit a steady increase in the number of branches over time, our researcher might choose those projects whose repository footprints exhibit different patterns. e.g., remain steady over time, alternate between a low and a high number of branches, remain at a constant number until the project’s half-point and then increases steadily.

While designing RepoGrams we were inspired by the promises of mining Git repositories as discussed by Bird et al. [13], such as ***Promise 2***: “*Git facilitates recovery of richer project history through commit, branch and merge information in the DAGs of multiple repositories.*” We use the abundance of data and meta-data recorded by Git to provide a rich framework for the various metrics that RepoGrams can visualize. We also took the perils they described into account when implementing specific metrics, some of which we detail in Section 4.3.

RepoGrams’s goal is to allow SE researchers to characterize and select a diverse set of evaluation targets, especially when their approach or perspective involves the analysis of project evolution. RepoGrams can help researchers strengthen claims to generality of their evaluated tool or method, and show that their tool does indeed support this diverse set of projects. RepoGrams also allows the research community to easily infer that the claims of past papers do indeed support a diverse set of projects by sending the repository URL of the software projects that were used to evaluate a paper to RepoGrams and inspecting the resulting repository footprints using relevant metrics.

## 1.1 Research questions

We developed RepoGrams to help SE researchers answer complex questions about repository histories to help them select evaluation targets. To help understand if RepoGrams serves this purpose, we posed the following research questions:

- **RQ1**: Can SE researchers use RepoGrams to *understand* and *compare* characteristics of a project’s source repository?

- **RQ2:** Will SE researchers consider using RepoGrams to select evaluation targets for experiments and case studies?

Before approaching the first two research questions we wanted to know if RepoGrams could be used by non-SE experts to investigate project histories, leading us to pose this third research question:

- **RQ3:** How usable is the RepoGrams visualization and tool?

During a user study we performed with SE researcher, many of them mentioned the need to define custom metrics in a visualization tool such as RepoGrams. We thus posed the following fourth and final research question:

- **RQ4:** How much effort is required to add metrics to RepoGrams?

We investigate these questions in Chapter 5.

## 1.2 Contributions

We make the following contributions:

- We report on a qualitative study of 55 papers from six major SE conferences in 2012–2014 to understand how SE researchers select project-based evaluation targets for their papers. Our results indicate that (1) most papers select projects based on qualitative metrics that are subjective, (2) most papers use 8 or fewer evaluation targets, and (3) there is a need in the SE research community for tool support in understanding project characteristics over time.
- Based on our qualitative study we designed and implemented RepoGrams, a web-based tool to analyze and juxtapose the evolution of one or more software projects. RepoGrams supports SE researchers in exploring the space of possible evaluation targets and is built on an extensible, metrics-based, visualization model that can be adapted to a variety of analyses. RepoGrams is free software, making it easily available for others to deploy and use.

- We evaluated RepoGrams with two user studies. We conducted a user study with 74 fourth year undergraduate students. We found that RepoGrams can be used by non-SE experts to investigate project histories. In another user study with 14 active SE researchers from the Mining Software Repositories (MSR) community we determined that they can use RepoGrams to understand and compare characteristics of a project's source repository.
- Finally, we evaluated the effort involved in adding six new metrics to RepoGrams in a case study with two software developers. We found that our study participants learned how to add their first new metric, and later add two more metrics, all within less than an hour for each metric.

The rest of this thesis is organized as follows. Chapter 2 lists related work. Chapter 3 describes the literature survey that we conducted. Chapter 4 discusses the design and implementation of RepoGrams. Chapter 5 describes the evaluation of RepoGrams with two user studies and a case study. Chapter 6 discusses ideas for future work, and finally Chapter 7 concludes the thesis. There are five appendices.

## Chapter 2

# Related work

In this section we present past work related to the one presented in the rest of this thesis. We split these works into categories: Section 2.1 lists works related to the methods that SE researchers employ when selecting evaluation targets for their empirical studies, Section 2.2 presents the literature survey which we discuss in Chapter 3, and finally Section 2.3 lists some of the relevant related work in the vast field of software visualizations.

### 2.1 Selection of evaluation targets

The problem of helping SE researchers perform better empirical evaluations has been previously considered from three different angles. First, there are ongoing efforts to curate high-quality evaluation targets in the form of bug reports [8, 51], faults [36], source code [60], etc. Such database artifacts promote scientific repeatability and comparison between proposed techniques. Second, researchers have developed tools like GHTorrent [30], Lean GHTorrent [31], Boa [24], MetricMiner [56], and the Orion search engine [15] to simplify access to, filtering, and analysis of projects hosted in large repositories such as GitHub. These tools make it easier to carry out evaluations across a large number of projects. Finally, recent work by Nagappan et al. has proposed improvements for sampling projects [45]. Their notion of sample coverage is a metrics-based approach for selecting and re-

porting the diversity of a sample set of SE projects. Unlike these three strands of prior work, we designed RepoGrams to support researchers in better decision making when it comes to selecting evaluation targets.

RepoGrams is designed to assist with research that evaluates software targets. However, SE researchers perform studies spanning the entire life-cycle of the SE process, using other types of data sources. For example, de Mello et al. [21] propose a framework for a more rigorous selection strategy when choosing human subjects for user studies, such as interviews or developer surveys.

## 2.2 Literature surveys

The work closest in methodology to our literature survey in Chapter 3 (albeit with different goals) is the literature survey by Hemmati et al. [33] which considers 117 research papers from MSR 2004–2012. They perform a grounded theory analysis to identify best practices prevalent in the MSR community across four high-level themes, including data acquisition and replication. Borges et al. [16] conducted a study of 891 papers from SE venues to discover which mechanisms are applied by researchers to support their empirical studies by reviewing the text and citations of each paper.

Collberg et al. [50] explored the repeatability of the research described in 601 papers from Association for Computing Machinery (ACM) conferences and journals, focusing on research that was backed by code. Ghezzi et al. [27] developed SOFAS, a platform devised to provide software analyses and to combine them into analysis workflows, aimed primarily at the MSR community in order to create replicable empirical experiments. To motivate their work they surveyed 88 papers that described empirical studies from MSR 2004–2011 conference. Siegmund et al. [55] performed a literature survey of 405 papers from SE conferences to get an overview of the current state of the art regarding discussions about validity and threats to validity in papers which include empirical research. Our literature review differs in its focus on *how* and *why* SE researchers selected the evaluation targets in their studies.

Jedlitschka et al. [35] review past guidelines for reporting study results in SE papers and propose the adoption of a unified standard. Among other matters, they



discuss the need to report on the sampling strategy that was used to select the experiment’s subjects. Sampling strategies are the main feature which we extracted in our literature survey.

## 2.3 Software visualizations

RepoGrams builds on a broad set of prior work on visualization of software evolution [23] and software metrics [41]. The focus of prior visualization work is on novel visualizations to support developers, managers, and other stakeholders in the software development process. The target population in our work on RepoGrams is the SE researcher community. We now overview the most relevant work from this space.

Novel visualizations span a broad range: Revision Towers [59] presents a visual tower that captures file histories and helps to correlate activity between files. GEVOL [18] is a graph-based tool for capturing the detailed structure of a program, correlating it against project activity and showing how it evolves over time. Chronos [54] assists developers in tracking the changes to specific code snippets across the evolution of a program. RelVis [48] visualizes multivariate release history data using a view based on Kiviat diagrams. The evolution matrix [38] supports multiple metrics and shows the evolution of system components over time. Chronia [28] is a tool to visualize the evolution of code ownership in a repository. Spectographs [66] shows how components of a project evolve over time and like RepoGrams extensively uses color. Other approaches to visualizing software evolution are further reviewed in [20]. Other such tools are surveyed by Storey et al. [57] in their work to define a framework for visualization tools that support awareness of human activities in software development. Some key features that differentiate RepoGrams from this rich prior work are its focus on commit granularity, avoidance of assumptions about the repository contents, support for comparison of multiple projects across multiple metrics, and a highly compact representation.

A more recent effort, The Small Project Observatory [39], visualizes ecosystems of projects. Complicity [46] is another web-tool that focuses on visualizing various metrics on software ecosystems, allowing users to drill down to a single project repository or observe the entire ecosystem. The tool’s main view visualizes

ecosystems on a scatter plot, and the users can set various metrics on five different dimensions: x/y coordinates, width/height or box, and color of box. RepoGrams differs from these two works in its emphasis on a single unifying view for all metrics and a focus on supporting SE researchers.

Another set of related work proposes tools to visualize certain software metrics. Some of these metrics are similar to the ones we have implemented in RepoGrams (Table 4.1). In contrast with this prior work our goal with RepoGrams is to support SE researchers. Additionally, RepoGrams is designed to support multiple metrics and to unify them within a single repository footprint visualization abstraction.

ConcernLines [61] is a tool to visualize the temporal pattern of co-occurring software concerns. Similarly to RepoGrams it plots the magnitude of concerns on a timeline and uses color to distinguish high and low concern periods. RepoGrams can be extended with a metric that counts concerns expressed in commit messages or in code comments. Fractal Figures [19], arguably the tool most visually similar to RepoGrams, visualizes commit authors from software repositories in Concurrent Versions System (CVS), using either one of two abstractions. RepoGrams's repository footprint abstraction is similar to Fractal Figures' abstraction called TimeLine View, which assigns a unique color to each commit author and lays all commits as colored squares on horizontal lines. Similarly to RepoGrams, each horizontal line represents a single software repository, and progression from left to right represents the passage of time. RepoGrams includes support for multiple metrics based on the artifacts exposed by the source repository, and we included a metric that assigns a unique color per author.

Visualization techniques in other domains bear relevance to RepoGrams. Chromograms [65] and history flow [63] are technique to visualize editor activity in Wikipedia. Chromograms uses color blocks arranged in a sequence but differs in its focus on a single metric to map text into colors. History flow resembles Seesoft [25] with the additional dimension of time. Begole et al. use *actograms* to visualize human computer activity [11]. In actograms a sequence of colored blocks denote activity over time. However, actograms capture activity over real-time while RepoGrams's focus is on capturing the sequence, and actograms is not designed for studying software activity.

## Chapter 3

# Project selection approaches in SE literature

To understand how SE researchers select software projects to act as the evaluation targets for their studies, we conducted a survey of recent papers published in SE venues. We used a grounded theory approach, which is an inductive methodology to construct theories through the analysis of data [58]. This chapter is organized as follows: in Section 3.1 we describe the protocol that we followed during this literature survey. In Section 3.2 we describe the final version of the codebook that we developed, in Section 3.3 we summarize the results. The full protocol, codebook, and raw results are listed in Appendix A.

### 3.1 Protocol

We started by reviewing a subset of research papers from large SE conferences that include research tracks where authors describe tools or methods that operate on software projects. We chose 30 and 29 random research-track papers from ICSE 2014 and MSR 2014, respectively. In reading these papers, we considered the selection strategy employed by the authors of each paper for choosing evaluation targets (if any) based on the text in the paper. We then iteratively derived a codebook (a set of codes, or categories) to characterize these strategies. Further, we considered the number of evaluation targets and which artifacts were studied by these papers.

**Table 3.1:** SE conferences that we reviewed in our literature survey.

Conference	Year	# papers (of 55)
Foundations of Software Engineering (FSE)	2012	10
Mining Software Repositories (MSR)	2012	5
Automated Software Engineering (ASE)	2013	5
Empirical Software Engineering and Measurement (ESEM)	2013	5
International Conference on Software Engineering (ICSE)	2013	10
Mining Software Repositories (MSR)	2013	10
Foundations of Software Engineering (FSE)	2014	10

Next, using this initial codebook, three more researchers from University of British Columbia (UBC) joined to iterate on the codebook by reading and discussing a random set of 55 research track papers from six SE conference proceedings in the years 2012–2014 (either five or ten papers from each conference). The surveyed conferences are summarized in Table 3.1.

We met several times to discuss between five and ten papers from one or two conferences at each meeting. Our discussions frequently caused us to update the codebook. By the end of each meeting, we derived a consensus on the set of codes for the discussed papers. As part of the meeting we also re-coded the previously discussed papers when changes to the codebook required doing so.

## 3.2 Codebook

The final version of the codebook includes the following five dimensions. We coded each paper in each of these five dimensions with the exceptions of papers that received the **IRR** code in the *selection criteria* dimension, as described below.

- *Selection criteria.* The type of criteria that the authors used to select projects for evaluation targets in a given paper. The resulting codes from this dimension are summarized in Table 3.2). For example, the code **DEV** stands for “*Some quality of the development practice was required from the selected evaluation targets. This quality does not necessarily have to be a unique feature, it could be something common such as the existence of certain data sets, usage of various aid tools such as an issue trackers, etc.*” Hence we

applied this code to those papers whose authors explained that the development process of the evaluation targets in the described research exhibited a particular process or used a particular set of tools that the authors deemed necessary for their evaluation.

- *Project visibility.* Captures the availability of project data, particularly the availability of the project artifacts, that were used in the paper (e.g., whether it is available online as open source, or is restricted to researchers through an industrial partner). The resulting codes from this dimension are summarized in Table 3.3.
- *Analyzes features over time.* A binary dimension (*yes* or *no*) to determine whether the authors analyzed project features over time or whether a single static snapshot of the project data was used.
- *Number of evaluation targets.* The number of distinct evaluation targets used in the paper’s evaluation. Note that we recorded the number of targets that the authors *claim* to evaluate as some targets can be considered to be a single project or many projects. For example, Android is an operating system with many sub-projects: one paper can evaluate Android as a single target, while another paper can evaluate the many sub-projects in Android.
- *Evaluated artifacts keywords.* Encodes the artifacts of the evaluation targets used in the paper’s evaluation (e.g., source code, issues in an issue tracking system, runtime logs).

In our study we found multiple cases in which more than one selection criteria code was applicable. For example, some papers relied on two distinct sets of projects (e.g., one set of projects was used as training data for some tool while another set of projects was used for evaluating that same tool). We therefore allowed multiple selection criteria codes for one paper. Table 3.2 lists the selection criteria codes, and the number of times each code was applied in the set of 55 papers that the four co-authors coded.

**Table 3.2:** Selection criteria codes and frequencies from our literature survey.

<b>Code</b>	<b>Description</b>	<b># papers (of 55*)</b>
<b>QUA</b>	The authors used informal qualities of the evaluation targets in their selection process. e.g., qualities such as age, code-base size, team composition, etc. The qualities are not defined strictly and there is no obvious way to apply a yes/no question that determines whether a new evaluation target would fit the selection criteria.	18
	Example: <i>“We have analyzed applications using widely-used components (such as the IE browser components and the Flash Player) and evaluated how our chosen reference programs and test subjects differ in terms of policy configurations under various workloads. Table 1 gives the detailed information on the analysis of the IE browser components”</i> [37]	
<b>DEV</b>	Some quality of the development practice was required from the selected evaluation targets. This quality does not necessarily have to be a unique feature, it could be something common such as the existence of certain data sets, usage of various aid tools such as an issue trackers, etc.	17
	Example: <i>“For this study we extracted the Jira issues from the XML report available on the Apache Software Foundation’s project website for each of the projects.”</i> [49]	
<b>REF</b>	References an existing and specific source of evaluation targets, such as another paper that has evaluated a similar technique/tool on a repository.	17
	Example: <i>“We evaluate our technique on the same search gold set used by Shepherd et al.”</i> [68]	

Continued on next page...

... Continued from previous page

<b>Code</b>	<b>Description</b>	<b># papers (of 55*)</b>
<b>DIV</b>	The authors mention diversity, perhaps not by name, as one of the features of the selected evaluation targets.	9
	Example: <i>“In this study, we analyze [...] three software systems [...] [that] belong to different domains”</i> [67]	
<b>ACC</b>	The authors had unique access to the evaluation targets, such as a software that is internal to the researching company. Not always explicit but sometimes implied from the text.	2
	Example: <i>“The case organization had been developing a telecommunications software system for over ten years. They had begun their transformation from a waterfall-like plan-driven process to an agile process in 2009.”</i> [32]	
<b>MET</b>	Random or manual selection based on a set of well-defined metrics. There is a well defined method to decide whether a new given project would fit the selection criteria. <b>MET</b> can be used for selection artifacts, but it must also provide constraints that also (perhaps implicitly) select the projects.	1
	Example: <i>“... we created a sample of highly discussed pull requests ... We defined ”highly discussed” as pull requests where the number of comments is one standard deviation (6.7) higher than the mean (2.6) in the dataset, filtering out all pull requests with less than 9 comments in the discussion.”</i> [62]	
<b>UNK</b>	Papers that do not provide an explanation of the selection process. This code is exclusive, and cannot be applied to the same set of evaluation targets if other codes were applied to that set.	2

Continued on next page...

...Continued from previous page

<b>Code</b>	<b>Description</b>	<b># papers (of 55*)</b>
<b>IRR</b>	Papers that are irrelevant to our focus: evaluation does not use projects or does not analyze repository information. This code is exclusive, and cannot be applied to a paper if other codes were applied to that paper.	15

\* Number of papers does not add up to 55 since multiple codes can be applied to each paper.

### 3.3 Results

The raw results of this literature survey are listed in the Appendix at Section A.3. We proceed to summarize these results.

Among the 55 papers coded by all four researchers, we used a code other than **IRR** on 40 (73%) papers. In the rest of this report we consider these 40 relevant papers as our global set.

Based on Table 3.2 we find that the three top selection criteria codes — **QUA**, **DEV**, and **REF** — had almost identical frequency at 18, 17, and 17 papers each (45%, 43%, and 43% respectively). That is, to select their evaluating targets the SE papers we considered relied on (1) qualitative aspects of the projects, (2) particular development practices, and (3) targets from previously published research. We found that 28 papers (70%) were coded with **QUA** and/or **DEV**. These two codes show that the majority of authors perform an ad-hoc selection of evaluation targets. When analyzing which artifacts were evaluated we found that 21 (53%) of the 40 papers evaluated the targets' source code or related artifacts such as patches or code clones. We propose that a tool that assists authors with the selection process of their evaluation target should inquire into informal metrics on both source code related artifacts of the projects themselves, and on artifacts relating to the development process of the projects.

Based on Table 3.3 we see that the vast majority of authors, at 36 papers (90%), prefer to run their evaluation on publicly available artifacts, such as the source code of open source projects. Industrial collaborations are a minority at 5 papers (13%).



**Table 3.3:** Project visibility codes and frequencies from our literature survey.

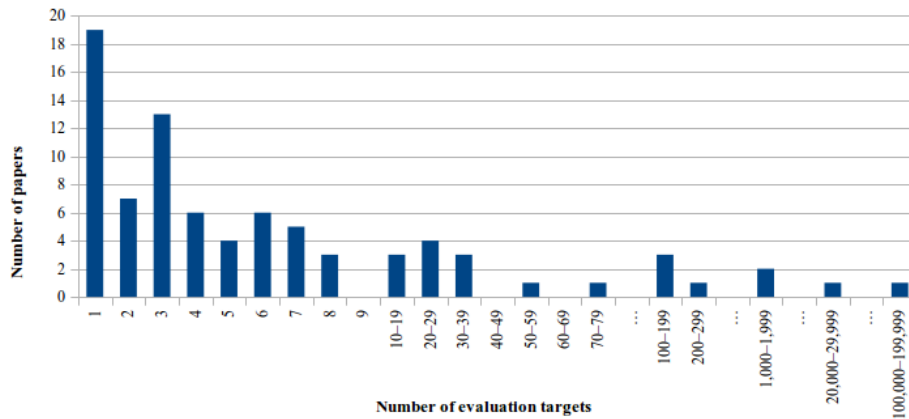
Code	Description	# papers (of 55)*
<b>PUB</b>	Projects were selected from a publicly available repository. Most likely open source, but not necessarily. Others can download the source code, binary, and/or data and run the evaluation themselves.	36
	Example: <i>“In this work, we analyze clone genealogies containing Type-1, Type-2, and Type-3 clones, extracted from three large open source software systems written in JAVA, i.e., ARGOURL, APACHE-ANT, and JBOSS.”</i> [67]	
<b>IND</b>	Industrial/company project. A collaboration with an industrial partner where the authors use their project. e.g., a company that performs research (e.g., Microsoft Research, Oracle Labs) and uses in-house projects. Can be an explicit mention of industrial partner (“we worked with Microsoft”) or a mention of a proprietary project.	5
	Example: <i>“In our work, we applied four unsupervised approaches [...] to the problem of summarization of bug reports on the dataset used in [34] (SDS) and one internal industrial project dataset (DB2-Bind)”</i> [40]	
<b>CON</b>	Project that the authors have complete control over, e.g., a new project from scratch solely for the purpose of the study, student projects.	2
	Example: <i>“We conducted three different development projects with undergraduate students of different duration and number of participating students.”</i> [22]	
<b>UNK</b>	When no details on the projects’ visibility was given	1
<b>IRR</b>	When the <i>selection criteria</i> is <b>IRR</b>	15

\* Number of papers does not add up to 55 since multiple codes can be applied to each paper.

There can be several reasons why most authors choose to run their evaluations on publicly available artifacts, e.g., reproducibility of the evaluation, the community’s familiarity with the evaluation target, ease of access to the data. Our tool should therefore focus on assisting authors in filtering potential evaluation targets out of vast repositories of public software projects, such as GitHub.

We also found that 16 papers (40%) analyzed their evaluation targets over time, indicating that many researchers are interested in studying changes over time and not just a snapshot. Our tool should have the capacity to consider temporal information about software projects.

**Figure 3.1:** Frequency of the number of evaluation targets by number of papers



Finally, considering all 114 total surveyed papers (both in the initial seeding process of the codebook and the joint coding process), we found that 84 papers (74%) performed an empirical evaluation on some artifacts of software projects (i.e., non-**IRR**), and 63 of these 84 papers (75%) evaluated their work with 8 or fewer evaluation targets. See Figure 3.1<sup>1</sup> for the frequency of the number of eval-

<sup>1</sup>For one paper it was unclear whether the authors evaluated 2 or 3 targets. In this chart it was counted as 3. One paper was not included in this chart since it neglected to mention the final number of evaluation targets.

uation targets by number of papers. Evaluations of entire large datasets are not as common as evaluations involving only a handful of targets. Our tool should support authors in small scale evaluations.

As mentioned in Chapter 2, some tools exist that are designed to assist researchers with the selection of large-scale sets of evaluation targets (i.e., hundreds or even hundreds of thousands), such as GHTorrent [30]. However, we found that the vast majority of SE researchers prefer to evaluate their work on small, manually curated sets. One might wonder why most SE researchers prefer working with small sets of evaluation targets. We found no direct answer to that question during our literature survey. However, having read these 55 papers we came up with the following two hypotheses: (1) in most papers we found that the authors require an in-depth analysis of each target to answer their research questions. These analyses are often manual processes which would not be possible were they analyzing thousands of evaluation targets. These questions do not gain from quantity but rather from the quality of the analysis. Working with a larger set will often increase the amount of busywork that the authors perform. (2) testing for diversity is difficult as its measure depends on the chosen metrics. A set of 8 evaluation targets might be considered diverse according to hundreds of different metrics, and another set of thousands of evaluation targets might be considered to lack diversity according to those same metrics. Hence, adding more evaluation targets to the set will not necessarily increase the SE community's confidence that the evaluated tool or method satisfies validity or claims to generalizability. Both of these hypotheses deserves further study, such as a survey of SE researchers.

Overall through this literature study we found that the process by which SE researchers choose their evaluation targets is often haphazard and rarely described in detail. This process could be supported by a tool designed to help SE researchers characterize and select software repositories to use as evaluation targets. Such a tool could also help the broader SE community better understand the authors' rationale behind a particular choice of evaluation targets.

## Chapter 4

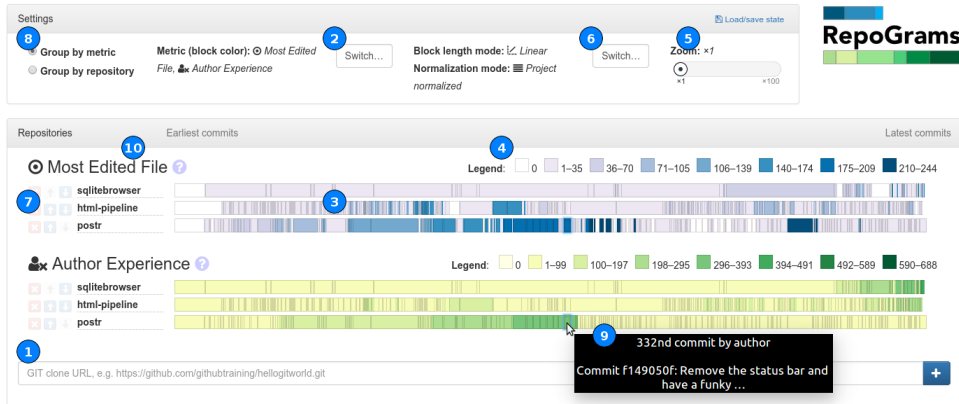
# RepoGrams's design and implementation

Based on the results of our literature survey we set out to create RepoGrams, a tool to understand and compare the evolution of multiple software repositories. RepoGrams is primarily intended to assist SE researchers in choosing evaluation targets before they conduct an evaluation of a tool or a method as part of their research projects. RepoGrams has three key features, each of which is grounded in our literature survey. First, RepoGrams is designed to support researchers in project selection. RepoGrams supports comparison of metrics for about a dozen projects (75% of papers evaluated their work with 8 or fewer evaluation targets). Second, it is designed to present multiple metrics side-by-side to help characterize the software development activity in a project overall (70% of papers used informal qualities to characterize their evaluation targets). Third, RepoGrams captures activity in project repositories over time (we found that 40% of papers consider software evolution in their evaluations) In the rest of this chapter we explain RepoGrams's design and implementation.

### 4.1 Design

We designed RepoGrams as a client-server web application, due to the convenience of use that such platforms provide to end users. Figure 4.1 shows a screenshot of a RepoGrams session with three added projects and two selected metrics.

**Figure 4.1:** RepoGrams interface: (1) input field to add new projects, (2) button to select the metric(s), (3) a *repository footprint* corresponding to a specific project/metric combination. The color of a *commit block* represents the value of the metric on that commit, (4) the legend for commit values in the selected metric(s), (5) zoom control, (6) button to switch block length representation and normalization mode, (7) buttons to remove or change the order of repository footprints, (8) way of switching between grouping by metric and grouping by project (see Figure 4.4), (9) Tooltip displaying the exact metric value and the commit message (truncated), (10) metric name and description



RepoGrams is designed to support the following workflow: the user starts by importing some number of project repositories. She does this by adding the projects' Git repository URLs to RepoGrams (① in Figure 4.1). The server clones<sup>1</sup> these Git repositories and computes metric values for all the commits across all of the repositories. Next, the user selects one or more metrics (② in Figure 4.1). This causes the server to transmit the precomputed metric values to the client to display. The metric values are assigned to colors and the interface presents the computed project repository footprints to the user (③ in Figure 4.1) along with the legend for each metric (④ in Figure 4.1).

<sup>1</sup>In Git nomenclature, "cloning" is the process of copying an entire Git repository with all its history and meta-data to the local machine.

RepoGrams currently requires that researchers manually add repositories that they already know, and base their selection on those. We discuss one idea to overcome this limitation in Section 6.1.

#### 4.1.1 Visual abstractions

We designed several visual abstractions to support tasks in RepoGrams, these are:

- **Repository footprint.** RepoGrams visualizes one or more metrics over the commits of one or more project repositories as a continuous horizontal line that we call a *repository footprint*, or footprint for short. (Figure 4.1 shows six repository footprints, two for each of the three project repositories). The footprints are displayed in a stack to facilitate comparison between project-s/metrics. A footprint is composed of a sequence of *commit blocks*. RepoGrams serializes the commits across all branches of a repository into a footprint using the commits' timestamps.
- **Commit block.** Each individual commit in the Git repositories is represented as a single block. The user selects a mode that determines what the width of the block will represent (see next bullet point). The metric value computed for a commit determines the block's color (see last bullet point).
- **Block width.** The length of a each commit block can be either a constant value, a linear representation of the number of LoC changed in the commit, or a logarithmic representation of the same. We also support two **normalization** variants:
  - *project normalized.* All lengths are normalized per project to utilize the full width available in the browser window. This mode prevents meaningful comparison between projects if the user is interested in contrasting absolute commit sizes. The footprints in Figure 4.1 use this mode
  - *globally normalized.* Block lengths are resized to be relatively comparable across projects.

All six possible combinations are demonstrated in Figure 4.2.

- **Block color.** A commit block’s color is determined by a mapping function that is defined in the metric’s implementation. This process is described in detail in Section 4.1.2.

**Figure 4.2:** All six combinations of block length and normalization modes

Modes		Repository footprints
Fixed	Globally	
	Project	
Linear	Globally	
	Project	
Logarithmic	Globally	
	Project	

Each cell contains two repository footprints that represent two artificially generated projects. The top repository footprint is of a repository with 6 commits in total, having 1, 2, 3, 4, 5, and 6 LoC changed. The bottom repository footprint is of a repository with 5 commits in total, having 1, 2, 4, 8, and 16 LoC changed.

### 4.1.2 Mapping values into colors with buckets

The computed values for each commit in each metric is mapped to a specific color with a buckets metaphor. For each metric we map several values together into a bucket as described below, and each bucket is assigned a color. Thus, the process of assigning a color for a commit block is to calculate the commit’s value in the metric, match that value to a bucket, and color the commit block based on the bucket’s matching color. The addition of a new repository to the view can cause some buckets to be recalculated, which will cause computed values to be reassigned and commit blocks to be repainted a different color.

A legend of each mapping created by the this process is displayed next to each selected metric (④ in Figure 4.1). For example, the second metric shown in Figure 4.1 is *author experience*. Using this example we can see that the most experienced author in the `sqlitebrowser` repository committed between 383–437

commits in this repository, as can be seen in the latest commits of that project (left-most commit blocks). In contrast, no author committed more than 218 commits in the `html-pipeline` repository, and no author committed more than 382 commits in the `postr` project.

**Figure 4.3:** Examples of legends generated from buckets for the *Languages in the Commit*, *Files Modified*, and *Number of Branches* metrics.

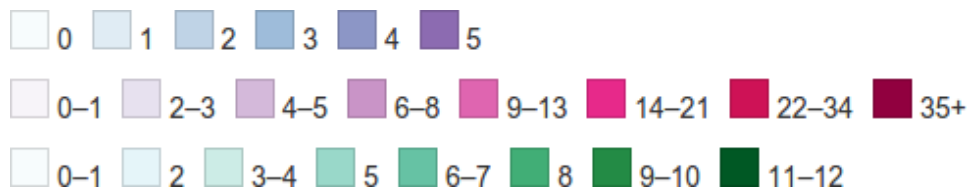


Figure 4.3 contains examples of three more legends, representing buckets that were generated from three metrics: *Languages in the Commit*, *Files Modified*, and *Number of Branches*. The buckets change automatically to match the repositories that were added by the user. RepoGrams currently supports three types of buckets:

- *fixed buckets* the metric has 8 buckets of predefined ranges. For example, the *commit message length* metric uses this bucket type. Buckets of these type do not change when new repositories are added. In the case of the *commit message length* metric the bucket ranges are  $\langle [0-1], [2-3], [4-5], [6-8], [9-13], [14-21], [22-34], [35-\infty] \rangle$ . Thus, two commit having 4 and 50 words in their commit message will be matched to the 3rd and 8th bucket, respectively, and their commit blocks will be colored accordingly.

An important benefit of fixed buckets is that adding or removing repositories from the view will not change the color of other repositories' commit blocks. On the other hand, outliers will be bundled together in the highest valued bucket. For example, a commit with a message length of 50 words will be bundled with a commit with a message length of 10,000 words due to the aforementioned fixed ranges.

The bucket colors for this type are ordered. They follow a linear progression such as increasing brightness on a single hue or a transition between two different hues.



- *uniform buckets* the metric has up to 8 buckets of equal or almost equal size, based on the largest computed value for that metric across all of the repositories. The buckets cannot always be of equal size due to integer division rules. For example, the *languages in the commit* metric uses this bucket type. If the highest value across all repositories is 7 or 12 then the bucket ranges are  $\langle \{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\} \rangle$  or  $\langle [0-1], [2-3], \{4\}, [5-6], [7-8], \{9\}, [10-11], [12-13] \rangle$ , respectively.

The distribution of ranges within these uniform buckets changes whenever the maximal value of a commit in the metric changes with the addition or removal of a repository to the view. Since the visualization is not stable the same color can represent one value at some point and another value after adding a new repository to the view. Whether this is an advantage or a disadvantage is up to the researcher. A more obvious disadvantage of this bucket type is that outliers can skew the entire visualization towards one extreme values. For example, if all commit values are within the range 1–10 except one commit with a value of 1,000, then using uniform buckets will cause all commits except the outlier to be placed in the lowest bucket, colored the same. We discuss potential solutions to this issue in Chapter 6.

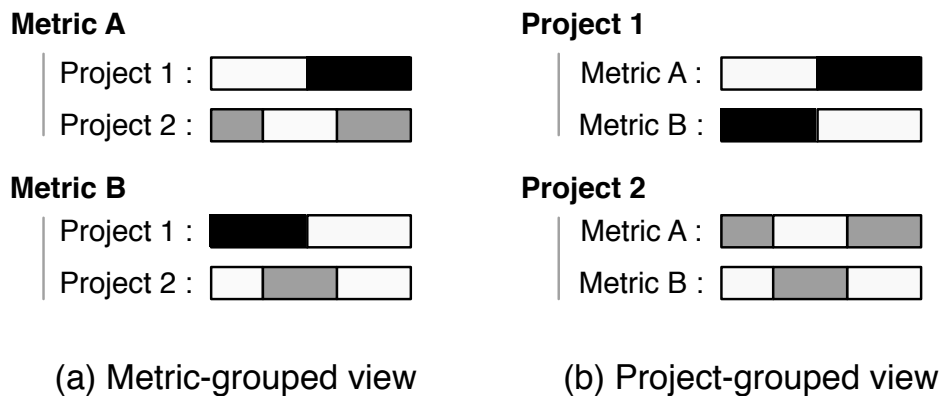
The bucket colors for this type are also ordered. Some metrics use a slightly modified version of this bucket type, such as having a separate bucket just for zero values and 7 equal/almost equal buckets on the remaining values, or starting from 1 instead of 0. These modifications are exposed in the legend.

- *discrete buckets* unlike the other two bucket types which deal with assigning numeric values from a linear or continuous progression of values to buckets, the discrete bucket types deal with discrete values. e.g., the *commit author* metrics assigns each unique commit author its own unique bucket.

The bucket colors for this type are categorical. Each bucket in this type has a completely different color to facilitate differentiation between the discrete values. The number of discriminable colors is relatively small, between six and twelve [43]. A metric that uses this type should limit the number of discrete values to twelve. This is not always possible. For example, a project repository might have hundreds of developers. Without a scheme to bundle

these developers together into shared buckets the *commit author* metric will have to display hundreds of colors. Solutions to this issue are metric-specific.

**Figure 4.4:** RepoGrams supports two ways of grouping repository footprints: (a) the metric-grouped view facilitates comparison of different projects along the same metric, and (b) the project-grouped view facilitates comparison of the same project along different metrics.



### 4.1.3 Supported interactions

The RepoGrams interface supports a number of interactions. The user can:

- **Scroll** the footprints left to right and **zoom in** and **out** (⑤ in Figure 4.1) to focus on either the entire timeline or a specific period in the projects' histories. In projects with hundreds or thousands of commits, some commit blocks might become too small to see. By allowing the user to zoom and scroll we enable them to drill down and explore the finer details of the visualization.
- **Change** the **block length mapping** and **normalization mode** (⑥ in Figure 4.1) as described in Section 4.1.1. The different modes emphasize different attributes, such as the number of commits or the relative size of each commit.

- **Remove** a project or **move** a repository footprint **up** or **down** (⑦ in Figure 4.1). By rearranging the repository footprints a user can visually derive ad-hoc groupings of the selected project repositories.
- **Change the footprint grouping** (⑧ in Figure 4.1) to group footprints by metric or by project (see Figure 4.4). The two modes can help the user focus on either comparisons of metrics within each project, or comparisons of projects within the same metric.
- **Hover** over or **click** on an individual commit block in a footprint to see the commit metric value, commit message, and link to the commit’s page on GitHub (⑨ in Figure 4.1). This opens a gateway for the user to explore the cause of various values, such as when a user is interested in understanding why a certain commit block has an outlier value in some selected metric.

## 4.2 Implementation details

RepoGrams is implemented as a client-server web application using a number of open source frameworks and libraries, most notably CherryPy [2] and Pygit2 [6] on the server side and AngularJS [1] on the client side. The server side is implemented mostly in Python, while the client side, as with all contemporary web application, is implemented in HTML5, CSS3, and JavaScript.

For convenience of deployment, RepoGrams can generate a Docker image that contains itself in a deployable format. Docker is an open platform for distributed applications for developers and system administrators [3] that enables rapid and consistent deployment of complex applications. By easing the deployment process we empower researchers who are interested in extending RepoGrams’s functionality to focus exclusively on their development efforts.

Each metric is implemented in two files. The first file is the server side implementation of the metric in Python. This file declares a single function, the name of which serves as the metric’s machine ID. The function takes one argument, a graph object that represents a Git repository, where each vertex in the graph represents a commit in that repository and contains commit artifacts. e.g., the commit log messages, the commit authors. The graph object has a method to iterate all the commit

nodes in temporal order. The function returns an ordered array containing the computed value for each commit in the temporal order of the commits. The second file is the client side implementation of the metric in JavaScript. This file declares meta-data about the metric: its name, description, icon, colors, and which mapper function the metric uses. It also defines a function to convert the raw computer value to human readable text to display in the tooltip (⑨ in Figure 4.1).

Some metrics might require a new mapper function. These are defined similarly by adding a single JavaScript file to the mapper directory in the application. A mapper is an object with two functions: `updateMappingInfo` and `map`. The function `updateMappingInfo` takes as argument an array with all the raw values returned from the server. It then calculates any changes to the buckets and returns true or false to indicate whether the buckets were modified at all. The function `map` takes as arguments a raw value as calculated by the server and a list of colors from the metric and returns the color that is associated with the equivalent bucket based on the work performed by `updateMappingInfo` earlier.

For a metric to be included and activated in a deployment of RepoGrams it must be registered in the Python base package file (`__init__.py`) of the metrics directory. By allowing deployers to modify which metrics are included we can support specific uses for RepoGrams that only require a subset of the existing metrics. Chapter 6 discusses an example of such a case for using RepoGrams as an educational tool.

### 4.3 Implemented metrics

As of this writing, RepoGrams has twelve built-in metrics. We list them in alphabetical order in Table 4.1 and describe them after the table. The Bucket column lists the type of mapping function used to assign a color to a metric value, as described earlier in Section 4.1.2. The Info column lists the type of information exposed in this metric. The metrics that we have developed so far can be categorized according to the kind of information they surface:

- *Artifacts* information. e.g., computed values using the source code
- *Development* process information. e.g., computed values about commit times
- *Social* information. e.g., who the commit author was

The Dev. column lists who developed this specific metric. This is either the original development team (**Team**) that developed RepoGrams’s earlier versions, or one of two developers (**Dev1** or **Dev2**) who developed metrics in a controlled settings to estimate the effort involved in adding a new metric to RepoGrams. This experiment is described in Section 5.3. The LoC column in the table represents the lines of code involved in the server-side calculation of a metric. Client-side code is not counted as it mostly consists of meta-data. The Time column lists the amount of time spent by a developer to add the metric. This was not counted for the original team since the development of these metrics was not conducted in a controlled setting.

**Table 4.1:** Alphabetical list of all metrics included in the current implementation of RepoGrams.

<b>Name</b>	<b>Bucket</b>	<b>Info</b>	<b>Dev.</b>	<b>LoC</b>	<b>Time</b>
Author Experience	Uniform	Social	Dev2	8	26 min
Branches Used	Discrete	Development	Team	5	—
Commit Age	Fixed	Development	Dev1	7	48 min
Commit Author	Discrete	Social	Dev1	34	52 min
Commit Localization	Fixed	Artifacts	Team	13	—
Commit Message Length	Fixed	Development	Team	6	—
Files Modified	Fixed	Artifacts	Dev2	6	42 min
Languages in a Commit	Uniform	Artifacts	Team	15	—
Merge Indicator	Uniform	Development	Dev2	5	44 min
Most Edited File	Uniform	Artifacts	Team	11	—
Number of Branches	Uniform	Development	Team	47	—
POM Files	Uniform	Artifacts	Dev1	6	30 min

It should be noted that these metrics are in no way comprehensive and usable for all researcher and research purposes. RepoGrams’s power comes not from its current set of metrics, but rather from its extensibility (as described in Section 4.2). Half of the metrics listed above were created during the experiment described in Section 5.3.

It is possible that these existing metrics will eventually create a bias among researchers regarding the selection process. However, as there is currently no tool designed for the same purpose as RepoGrams, we believe that the inclusion of these metrics is an improvement over the current state of the art in which researchers do not currently base their selection on evidence. Mitigating this potential bias remains a problem for future researchers.

We proceed to describe each metric. For each metric we also provide an example that exhibits why researchers might care about this metric.

- *Author Experience.* The number of commits a contributor has previously made to the repository. *For example, a researcher interested in studying developer seniority across software projects can use this metric to choose projects that exhibit different patterns of author experience. e.g., similar between developers vs. skewed for a minority of developers in the team.* This metric was added based on a suggestion by one of the participants in the SE researchers study. See Section 5.2.3
- *Branches Used.* Each implicit branch [13] is associated with a unique color. A commit is colored according to the branch it belongs to. *For example, a researcher interested in studying whether projects exhibit strong branch ownership by individual developers can correlate this metric with the Commit Author metric.*
- *Commit Age.* Elapsed time between a commit and its parent commit. For merge commits we consider the elapsed time between a commit and its youngest parent commit. *For example, a researcher interested in exploring whether a correlation exists between the elapsed time that separates commits, and the likelihood that the latter commit is a bug-introducing commit, can use this metric to select projects that contain different patterns of commit ages.*
- *Commit Author.* Each commit author is associated with a unique color. A commit block is colored according to its author. *For example, a researcher interested in studying the influence of dominant contributors on minor con-*

*tributors in open source projects can begin their exploration by using this metric to identify projects that exhibit a pattern of one or several dominant contributors.*

- *Commit Localization.* Fraction of the number of unique project directories containing files modified by the commit. Metric value of 1 means that all the modified files in a commit are in a single directory. Metric value of 0 means all the project directories contain a file modified by the commit. *For example, researchers interested in cross-cutting concerns could use this metric to search for projects to study. A project with many commits that have a low value of localization can potentially have a high level of cross-cutting concerns.*
- *Commit Message Length.* The number of words in a commit log message. *For example, a researcher interested in finding whether a correlation exists between the commit message lengths and developer experience can compare this metric with the Author Experience metric and select projects that exhibit different patterns for further study.*
- *Files Modified.* The number of files modified in a particular commit, includes new and deleted files. *For example, a researcher interested in studying project-wide refactoring operations can use this metric to find points in history where a large number of files were modified in a repository. A large number of files modified could potentially indicates that this event has occurred.* This metric was added based on a suggestion by one of the participants in the SE researchers study.
- *Languages in a Commit.* The number of unique programming languages used in a commit based on filenames. *For example, a researcher interested in studying the interaction between languages in different commits can use this metric to identify projects that have many multilingual commits.*
- *Merge Indicator.* Displays the number of parents involved in a commit. Two or more parents denote a merge commit. *For example, a researcher may be interested in studying projects with many merge commits. This metric can*

*reveal whether a project is an appropriate candidate for such a study.* This metric was added based on a suggestion by one of the participants in the SE researchers study.

- *Most Edited File.* The number of times that the most edited file in a commit has been previously modified. *For example, a researcher interested in studying “god files” can use this metric to identify projects where a small number of files have been edited multiple times over a short period. The existence of such files potentially indicates the existence of “god files”.*
- *Number of Branches.* The number of branches that are concurrently active at a commit point. *For example, a researcher interested in studying how and why development teams change the way they use branches can use this metric to identify different patterns of branch usage for further exploration.*
- *POM Files.* The number of POM files changed in every commit. *For example, a researcher interested in exploring the reasons for changes to the parameters of the build scripts of projects can use this metric to find points in history where those changes occurred.*

The *POM Files* metric is an example of a specific case that can be generalized. In this case, to highlight edits to files with a user-determined filename pattern. Customizable metrics are discussed as future work in Section 6.1



## Chapter 5

# Evaluation

We conducted two user studies and an experiment to answer the research questions we posed earlier in Section 1.1. This chapter describes these evaluations and discusses the results.

For convenience we repeat the research questions here. For a more detailed discussion of these research questions see Section 1.1:

- **RQ1:** Can SE researchers use RepoGrams to *understand* and *compare* characteristics of a project’s source repository?
- **RQ2:** Will SE researchers consider using RepoGrams to select evaluation targets for experiments and case studies?
- **RQ3:** How usable is the RepoGrams visualization and tool?
- **RQ4:** How much effort is required to add metrics to RepoGrams?

The rest of this chapter is organized as follows: Section 5.1 details a user study with undergraduate students that answers **RQ3**, Section 5.2 details a user study with SE researchers that answers **RQ1** and **RQ2**, and finally Section 5.3 details a case study that answers **RQ4**.

## 5.1 User study with undergraduate students

In this first evaluation, a user study with undergraduate students, we aimed to determine if individuals less experienced with repositories and repository analysis could comprehend the concept of a *repository footprint* and effectively use RepoGrams (RQ3). The study was conducted in a fourth year software engineering class: A total of 91 students participated and 74 students completed the study. Participation in the study in class was optional. We incentivized participation by raffling off five \$25 gift cards for the university's book store among the participants that completed the study.

### 5.1.1 Methodology

The study consisted of two parts: a 10 minute lecture demonstrating RepoGrams, and a 40 minute web-based questionnaire. The questionnaire asked the participants to perform tasks with RepoGrams and answer questions about their perception of the information presented by the tool.

The questionnaire had three sections<sup>1</sup>: (1) a *demographics* section to evaluate the participants' knowledge and experience, (2) four *warm-up questions* to introduce participants to RepoGrams and (3) ten *main questions* in three categories:

- *Metric comprehension.* Six questions to test if participants understood the meanings of various metrics.
- *Comparisons across projects.* Three questions to test if the participants could recognize patterns across repository footprints to compare and contrast projects and to find positive or negative correlations between them.
- *Exploratory question.* One question to test whether participants could translate a high-level question into tasks in RepoGrams.

Before each of the main questions, participants had to change selected metrics and/or block length modes. A detailed explanation on the metrics used in each question was provided.

---

<sup>1</sup>The full questionnaire is listed in Appendix B.

The questions were posed in the context of 5 repositories selected from 10 random projects from GitHub’s trending projects page that were open source and had up to 1,500 commits. From those 10 projects we systematically attempted permutations of 5 projects<sup>2</sup> until we found a permutation such that all 5 repository footprints fit the ten main questions from the study. We established ground truth answers for each question. The final set of project repositories in the study had a min / median / max commit counts of 581 / 951 / 1,118, respectively.

### 5.1.2 Results

We received 74 completed questionnaires from the 91 participants. These 74 participants answered a median of 8 of the 10 answers correctly. The median time to complete a *metric comprehension* question was 1:20 min, *comparison across projects* was 1:32 min, and the *exploratory question* was 2:51 min. In total, participants took on median 14:10 min to answer the main questions. The success with which participants answered questions in relatively short time provides evidence that RepoGrams is usable by a broad population (**RQ3**). Interestingly, we found no significant correlation between a participants’ success rate and their industry or Version Control System (VCS) experience.

To provide more insight into how these users fared, we highlight the results for two questions; we do not discuss the results of the other eight questions in detail.

**Question 5**, is an example of a *metric comprehension* question that asked: “Using the Languages in a Commit *metric and any block length, which project is likely to contain source code written in the most diverse number of different languages?*” (94% success rate)

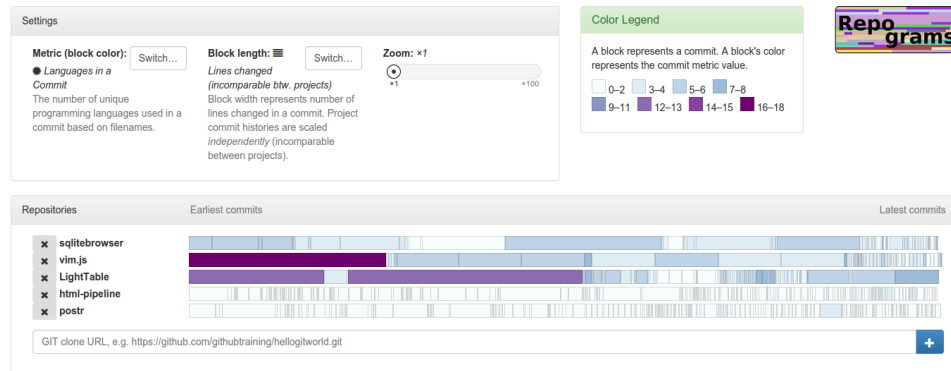
In this task the participants were shown 5 repository footprints, as seen in Figure 5.1. Our ground truth consisted of two answers that were visually similar: (1) a footprint that had one commit block in the 16–18 range (chosen by 48 (72%) of participants), and (2) a footprint that had two commits block in the 14–15 range (chosen by 15 (22%) of participants). The remaining three footprints had all their commit blocks in the 5–6 range or lesser. The high success rate for this question in-

---

<sup>2</sup>A 6th project was later taken at random. Its repository footprint was to be removed by the participants at the beginning of the study as part of a task intended to familiarize the participants with the interface.

indicates that the users were able to comprehend the metric presented by RepoGrams and to find patterns and trends based on the repository footprints of projects.

**Figure 5.1:** RepoGrams showing the repository footprints as it was during the user study with undergraduate students, **question 5**.



**Question 12** is an example of a *comparisons across projects* question: “Using the languages in a commit metric and the fixed block length, which two project repositories appear to have the most similar development process with each other?” (81% success rate)

In this question, we asked the participants to explain their choice of the two repositories. We then coded the answers based on the attributes the participants used in their decision. For each question we created at least two codes, one code indicates that the explanation was focused on the metric values (e.g., “These two projects stick to at most 2 languages at all times in their commits. Sometimes, but rarely, they use 3–4 languages as indicated by the commits”), the other code indicates that the explanation was focused on the visualization (e.g., “The shading in both projects were very light”). Occasionally an explanation would discuss both the metric values and the visualization (e.g., “It seems that both languages use a small number of languages throughout the timeline, since colors used for those projects are mainly light”), in which case we applied both codes. When another visual or abstract aspect was discussed in the participant’s explanation we created codes to match them.

We found that the participants who discussed the meaning of the metric values had a higher success rate (65%) compared to those participants who relied solely on the visualization (27%). A similar trend is apparent in other question where we asked the participants to explain their answer.

### **5.1.3 Summary**

This user study on individuals with less SE training indicates that RepoGrams can be used by a broader population of academics with a computer science background. However, when individuals rely on the visualization without an understanding of the metric underlying the visualization, mis-interpretation of the data may occur.

## **5.2 User study with SE researchers**

To investigate the first two research questions (**RQ1** and **RQ2**), we performed a user study with researchers from the SE community. This study incorporated two parts: first, participants used RepoGrams to answer questions about individual projects and comparisons between projects; second, participants were interviewed about RepoGrams. We recruited participants for the study from a subset of authors from the MSR 2014 conference, as these authors likely performed research involving empirical studies using software projects as evaluation targets, and many have experience with repository information. These authors are the kind of SE researchers that might benefit from a tool such as RepoGrams. Some of the authors forwarded the invitation to their students whom we included in the study.

We used the results of the previous user study and the comments given by its participants to improve the tool prior to running this user study with SE researchers. For example, in the first study RepoGrams only supported the display of one metric at a time. Participant comments prompted us to add support for displaying multiple metrics. We also realized that some labels and descriptions caused confusion and ambiguity, we endeavored to clarify their meanings. On the technical side, we found that due to the server load during the study, performance was a recurring complaint. We made significant improvements to make all actions in the tool faster.

The study had 14 participants: 5 faculty, 1 post doc, 6 PhD students, and 2 masters students. Participants were affiliated with institutions from North Amer-

ica, South America, and Europe. All participants have research experience analyzing the evolution of software projects and/or evaluating tools using artifacts from software projects.

Similarly to the undergraduate study, we raffled off one \$100 gift card to incentivize participation. The study was performed in one-on-one sessions with each participant: 5 participants were co-located with the investigator and 9 sessions were performed over video chat.

### 5.2.1 Methodology

Each session in the study began with a short demonstration of RepoGrams by the investigator, and with gathering demographic information. A participant then worked through nine questions presented through a web-based questionnaire.<sup>3</sup>

The first three questions on the questionnaire were aimed at helping a participant understand the user interface and various metrics (5 minutes limit for all three questions). Our intent was to ensure each participant gained a similar level of experience with the tool prior to the main questions.







The remaining six questions tested whether a participant could use RepoGrams to find advanced patterns. Questions in this section were of the form “group the repositories into two sets based on a feature”, where the feature was implied by the chosen metric (3–7 minutes limit per question). Table 5.1 lists these questions in detail. We then interviewed each participant in a semi-structured interview described in Section 5.2.3.

For the study we chose the top 25 trending projects (pulled on February 3rd, 2015) for each of the ten most popular languages on GitHub [64]. From this set we systematically generated random permutations of 1–9 projects for each question until we found a set of projects such that the set’s repository footprints fit the intended purpose of the questions. The final set of project repositories in the study had a min / median / max commit counts of 128 / 216 / 906, respectively.

---

<sup>3</sup>The full questionnaire is listed in Appendix C.

**Table 5.1:** Main questions from the advanced user study.

#	Question	# repository footprints	Dist.
4	Which of the following statements is true? <i>There is a general {upwards / constant / downwards} trend to the metric values.</i>	1	
5	Categorize the projects into two clusters: (a) projects that use Maven (include .pom files), (b) projects that do not use Maven.	9	
6	Categorize the projects into two clusters: (a) projects that used a single master branch before branching off to multiple branches, (b) projects that branched off early in their development.	5	
7	Categorize the projects into two clusters: (a) projects that have a correlation between branches and authors, (b) projects that do not exhibit this correlation.	8	
8	Categorize the projects into two clusters: (a) projects that have one dominant contributor, based on number of <b>lines of code changed</b> , (b) projects that do not have such a contributor. A dominant contributor is one who committed at least 50% of the LoC changes in the project.	3	
9	<i>Same as 5, with number of <b>commits</b> instead of number of <b>lines of code changed</b>.</i>	3	

### 5.2.2 Results

To give an overall sense of whether SE researchers were in agreement about the posed questions, we use a graphic in the Dist. column of Table 5.1. In this column, each participant's answer is represented by a block; blocks of the same color denote identical answers. For example, for question 6, twelve participants chose one answer and two participant chose a different answer each; a total of three distinct answers to that question.

The Dist. column of Table 5.1 shows widespread agreement amongst the researchers for questions 4 and 5. These questions are largely related to interpreting metrics for a project. This quantitative agreement lends support to the *under-*

*standing* part of **RQ1**. More variance in the answers resulted from the remaining questions that target the *comparison* part of **RQ1**; these questions required more interpretation of metrics and comparisons amongst projects.

To gain more insight into the SE researchers use of RepoGrams, we discuss each of the main questions.

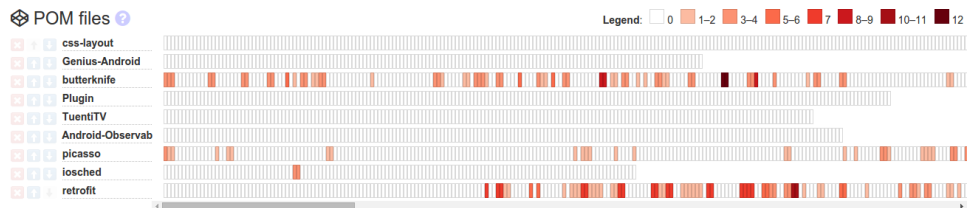
**Question 4** asked the participants to recognize a trend in the metric value in a single repository. The majority of participants (12 of 14) managed to recognize the trend almost immediately by observing the visualization.

**Figure 5.2:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 4**.



**Question 5** asked the participants to identify repositories that have a non-zero value in one metric. The participants considered 9 repository footprints where the metric was *POM Files*: a value of  $n$  indicates that  $n$  POM files were modified in a commit. This metric is useful for quickly identifying projects that use the Maven build system [4]. All except one participant agreed on the choice for the nine repositories. This question indicates that RepoGrams is useful in distinguishing repository footprints that contain a common feature, represented by a particular color.

**Figure 5.3:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 5**.

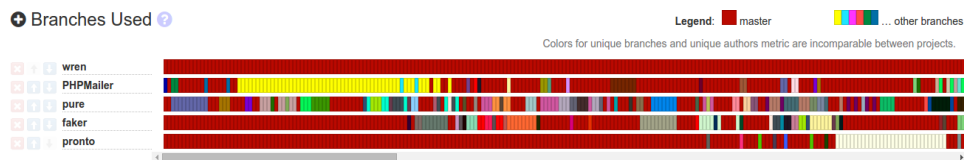




**Question 6** asked the participants to identify those repositories in which the repository footprints started with a sequence of commit blocks of a particular color. The participants considered 5 repository footprints. The metric was *Branches Used*: each branch is given a unique color, with a specific color reserved for commits to the master branch. All five footprints contained hundreds of colors.

The existence of a leading sequence of commit blocks of a single color in a *Branches Used* metric footprint indicates that the project used a single branch at the start of its timeline or that the project was imported from a centralized version control system to Git. All participants agreed on two of the footprints and all but one agreed on each of the other footprints. This indicates that RepoGrams is useful in finding long sequences of colors, even within footprints that contain hundreds of colors.

**Figure 5.4:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 6**.

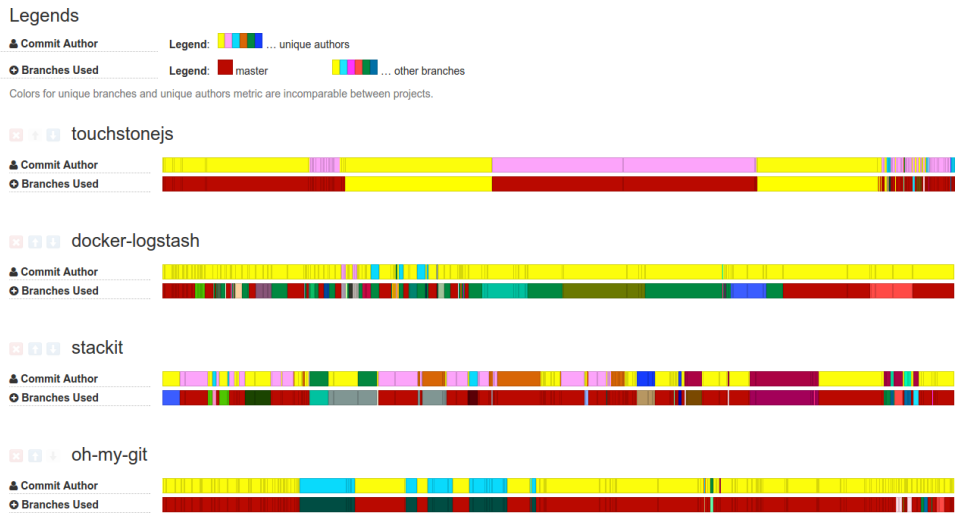


**Question 7** asked the participants to identify those repositories in which the repository footprints for two metrics contained a correspondence between the colors of the same commit block. The participants considered a total of 8 repository footprints, with two metrics for four project. The two metrics were *Commit Author* and *Branches Used*. A match in colors between these two metrics would indicate that committers in the project follow the practice of having one branch per author. This is useful to identify for those studies that consider code ownership or the impact of committer diversity on feature development [14].

In the task the number of colors in a pair of footprints for the same repository ranged from a few (<10) to many (>20). The majority (twelve) of participants agreed on their choices for the first, second, and fourth repository pairs. But, we found that they were about evenly split on the third repository (eight vs. six partic-

ipants). This indicates that RepoGrams is useful in finding a correlation between repository footprints when the number of colors is low, but it is less effective with many unique colors.

**Figure 5.5:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 7**.



**Question 8 and 9** asked the participants to estimate the magnitude of non-continuous regions of discrete values. The participants were relatively split on these results. We conclude that RepoGrams is not the ideal tool for performing this type of task.

**Figure 5.6:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 8**.



**Figure 5.7:** RepoGrams showing the repository footprints as it was during the user study with SE researchers, **question 9**.



### 5.2.3 Semi-structured interview

After the participants finished the main tasks, we conducted a semi-structured interview to discuss their experiences with RepoGrams. We asked 5 questions, and allotted a maximum of 10 minutes for this part. No interview lasted that long. The questions were:

- Do you see RepoGrams being integrated into your research/evaluation process? If so, can you give an example of a research project that you could use/could have used RepoGrams in?
- What are one or two metrics that you wish RepoGrams included that you would find useful in your research? How much time would you be willing to invest in order to write code to integrate a new metric?
- In your opinion, what are the best and worst parts of RepoGrams?
- Choose one of the main tasks that we asked you to perform. How would you have performed it without RepoGrams?
- Do you have any other questions or comments?

Since the interviews were mostly unstructured, participants went back and forth between questions when replying to our questions. Hence, the following summary of all interviews also takes an unstructured form:

Of the 14 participants, 11 noted that they want to use RepoGrams in their future research: “*I would use the tool to verify or at least to get some data on my selected projects*” [P12]<sup>4</sup> and “*I would use RepoGrams as an exploratory tool to see the characteristics of projects that I want to choose*” [P9]. They also shared past research projects in which RepoGrams could have assisted them in making a more informed decision while choosing or analyzing evaluation targets. The remaining 3 participants said that they do not see themselves using RepoGrams in their research but that either their students or their colleagues might benefit from the tool.

Most participants found the existing metrics useful: “*Sometimes I’m looking for active projects that change a lot, so these metrics [e.g., Commit Age] are very useful*” [P8]. However, they all suggested new metrics and mentioned that they would invest between 1 hour to 1 week to add their proposed metric to RepoGrams. In Section 5.3 we detail a case study in which we add three of these proposed metrics to RepoGrams and show that this takes less than an hour per metric. The proposed metrics ranged from simple metrics like counting the number of modified files in a commit, to complex metrics that rely on third-party services and tools. For example, two participants wanted to integrate tools to compute the complexity of a change-set based on their own prior works. Another participant wanted to integrate a method to detect the likelihood that a commit is a bug-introducing commit. Yet another participant suggested a metric to calculate the code coverage of the repository’s test suite to consider the evolution of a project’s test suite over time.

A few of the suggestions would require significant changes. For example, inspired by the *POM Files* metric, two participants suggested a generalized version of this metric that contains a query window to select a file name pattern. The metric would then count the number of files matching the query in each commit. We discuss this idea and others in Chapter 6.

The participants also found that RepoGrams helped them to identify general historical patterns and to compare projects: “*I can use RepoGrams to find general trends in projects*” [P3] and “*You can find similarities . . . it gives a nice overview*

---

<sup>4</sup>We use [P1]–[P14] to refer to the anonymous participants.

*for cross-projects comparisons*” [P13]. They also noted that RepoGrams would help them make stronger claims in their work: *“I think this tool would be useful if we wanted to claim generalizability of the results”* [P4].

One of our design goals was to support qualitative analysis of software repositories. However, multiple participants noted that the tool would be more useful if it exposed statistical information: *“It would help if I had numeric summaries.”* and *“When I ask an exact numeric question this tool is terrible for that. For aggregate summaries it’s not good enough”* [P6]

Another design limitation that bothered participants is the set temporal ordering of commits in the repository footprint abstraction: *“Sometimes I would like to order the commits by values, not by time”* [P7] and *“I would like to be able to remove the merge commits from the visualizations.”* [P14]. Related to this, a few participants noted the limitation that RepoGrams does not capture real time in the sequence of commit blocks: *“the interface doesn’t expose how much time has passed between commits, only their order.”* [P7]

The participants were asked to choose one of the tasks and explain how they would solve that task without using RepoGrams. Two generalized approaches emerged repeatedly. The most common approach was to write a custom script that clones the repositories and performs the analysis. One participant mentioned that their first solution script to solve task 6 (identifying projects that use or have used Maven) would potentially get wrong results since they intended to only observe the latest snapshot and not every commit from the repository. A software project might have used Maven early in its development and later switched to an alternative build system, in which case its latest snapshot would not contain POM files and the script would fail to recognize this repository.

Alternatively some participants said that they would import the meta-data of the Git repositories into a spreadsheet application and perform the analysis manually. Some participants mentioned that GitHub exposes some visualizations, such as a histogram of contributors for repositories. These visualizations are per-repository and do not facilitate comparisons.

### 5.2.4 Summary

This study shows that SE researchers can use RepoGrams to understand characteristics about a project's source repository and that they can, in a number of cases, use RepoGrams to compare repositories (**RQ1**), although the researchers noted areas for improvement. Through interviews, we determined that RepoGrams is of immediate use to today's researchers (**RQ2**) and that there is a need for custom-defined metrics.

### 5.3 Estimation of effort involved in adding new metrics

The SE researchers who participated in the user study described in the previous section had a strong interest in adding new metrics to RepoGrams. Because researchers tend to have unique research projects that they are interested in evaluating, it is likely that this interest is true of the broader SE community as well. In this last study we evaluated the effort in adding new metrics to RepoGrams (**RQ4**).

The metrics were implemented by two junior SE researchers: (Dev1) a masters student who is the author of this thesis, and (Dev2) a fourth year Computer Science undergraduate student. Dev1 was, at the time, not directly involved in the programming of the tool and was only slightly familiar with the codebase. Dev2 was unfamiliar with the project codebase. Each developer added three new metrics (bottom six rows in Table 4.1).

Dev1 added the *POM Files*, *Commit Author*, and *Commit Age* metrics. Prior to adding these metrics Dev1 spent 30 minutes setting up the environment and exploring the code. The *POM Files* metric took 30 minutes to implement and required changing 16 LoC<sup>5</sup>. Dev1 then spent 52 minutes and 48 minutes developing the *Commit Author* and *Commit Age* metrics, changing a similar amount of code for each metric.

Dev2 implemented three metrics based on some of the suggestions made by the SE researchers in Section 5.2.3: *Files Modified*, *Merge Indicator*, and *Author Experience*. Prior to adding these metrics Dev2 spent 39 minutes setting up the

---

<sup>5</sup>Note that these numbers are different from those listed in Table 4.1. See the closing paragraph in Section 5.3.1 for an explanation on this disparity.

environment and 40 minutes exploring the code. These metrics took 42, 44, and 26 minutes to implement, respectively. All metrics required changing fewer than 30 LoC.

### 5.3.1 Summary

The min / median / max times to implement the six metrics were 26 / 43 / 52 minutes. These values compare favorably with the time that it would take to write a custom script to extract metric values from a repository, an alternative practiced by almost all SE researchers in our user study. The key difference, however, is that by adding the new metric to RepoGrams the researcher gains two advantages: (1) the resulting project pattern for the metric can be juxtaposed against project patterns for all of the other metrics already present in the tool, and (2) the researcher can use all of the existing interaction capabilities in RepoGrams (changing block lengths, zooming, etc).

At the time of this case study, the architecture of the tool required that developers modify existing source code files in order to add a new metrics. While this complicated the process of adding a new metric, the experiment shows that developers can do so in less than 1 hour after an initial code exploration. We attempted to streamline this process even further by reworking the architecture of the tool to move the implementation of metrics to separate files as described in Section 4.2. During this architectural change we had to rewrite parts of the existing metrics. Table 4.1 lists the LoC count *after* this change. We also added documentation to assist developers in setting up their development environment and created examples that demonstrate how to add new metrics.

## Chapter 6

# Future work

In this section we discuss plans for future work involving RepoGrams. Some of these are in response to current limitations of tool, while others are new ideas aimed to expand the reach of this tool beyond its current SE researchers focus.

### 6.1 Additional features

*Studying populations of projects.* RepoGrams requires the user to add one project at a time. We are working to add support for importing random project samples from GitHub. RepoGrams can be integrated with a large database of repositories such as GHTorrent [31]. Users can then use a query language (such as SQL or a unique domain-specific language) to query by attributes that are recorded in the database. e.g., filter to select random projects that use a particular programming language, have a particular team size, specific range of activity in a period of time. By randomizing the selection based on strictly defined metrics such as these, SE researcher can have a stronger claim of generalizability in their papers.

*Supporting custom metrics.* SE researchers in our user study (Section 5.2) wanted more specialized metrics that were, unsurprisingly, related to their research interests. As mentioned in Section 5.2.3 we are working on a solution in which specific metrics can be customized in the front-end. These metrics will have parameters that can be set by the users, and calculated by the server for display. For example, the *POM Files* metric is a specific case of a more generic metric



that counts the number of modified files in a commit that match a specific pattern (e.g., \*.pom). We are also considering another solution in which a researcher could write a metric function in Python or a domain-specific language and submit it to the server through the browser. The server would integrate and use this user-defined metric to derive repository footprints. We plan to explore the challenges and benefits of this strategy.

*Supporting non-source-code historical information.* RepoGrams currently supports Git repositories. However, software projects may have bug trackers, mailing lists, Wikis, and other resources that it may be useful to study over time and compare with repository history in a RepoGrams interface. We plan to extend RepoGrams with this information by integrating with the GitHub API, taking into account concerns pointed out in prior work [13].

*Robust bucketing of metric values.* Uniform bucket sizing currently implemented in RepoGrams has several issues. For example, a single outlier metric value can cause the first bucket to become so large as to include most other values except the outlier. One solution is to generate buckets based on different distributions and to find outliers and place them in a special bucket. We will try different configurations and algorithms for bucketing, as well as enabling real-time modifications by users, in an attempt to solve this issue and other similar ones.

*Supporting collaborations.* We added an import/export feature that saves the local state in a file, which can be shared and then loaded into the tool by others. This is a preliminary solution to the problem of sharing the data sets and visualizations between different researchers working on the same project. We intend to design and implement a more contemporary solution. e.g., sharing a link to the current state instead of sharing files.

## 6.2 Expanded audience

*Educational study.* We are exploring, together with other researchers in our department, the option of using RepoGrams for educational purposes. We are designing two experiments that involves integrating RepoGrams in a third year SE class in which student teams develop a software project for the entire term. In one experiment we are attempting to find correlation between specific repository footprints

and final grades given to student teams in previous terms when the class was taught. In the other experiment we will integrate RepoGrams into the periodic evaluations of the student teams by the Teaching Assistants (TAs) during the term. In this experiment we are attempting to discover whether the visualizations shown by the tool help guide the student teams towards a more successful completion of the project and to better understand the expectations of their TA.

*Use of RepoGrams in the industry.* RepoGrams is designed for SE researchers. However, it is possible that it can be also used by other target audiences. One example is for managers or software developers in industry. They can use RepoGrams to track project activity and potentially gain insights about the development process.

### **6.3 Further evaluations**

*Long term benefits of RepoGrams for SE researchers.* Our evaluation did not conclusively show that RepoGrams helps SE researchers in selecting their evaluation targets. We plan to use RepoGrams in our own SE research work and to collect anecdotal evidence from other researchers to be able to eventually argue this point conclusively.

*Evaluating new features.* We added several new features to RepoGrams that we did not evaluate. One example of such a feature is the logarithmic block length mode described in Section 4.1.1. This block length mode was added after the user study with SE researchers (Section 5.2) and thus was not evaluated.

Another feature that we added to the tool was created while designing the above mentioned educational study. We found that many student groups perform large scale refactoring such as running a code formatter on specific commits or adding large third party libraries to their repositories. The commits blocks for these commits take up a sizable part of the repository footprint, yet they are of no interest in this study. We implemented a feature to hide individual commits from the view. Evaluating whether it is a useful feature for the SE community remains future work.

We intend to test this new feature and others as part of any future evaluation where they might prove relevant.

## Chapter 7

# Conclusion

The widespread availability of open source repositories has had significant impact on SE research. It is now possible for an empirical study to consider hundreds of projects with thousands of commits, hundreds of authors, and millions of lines of code. Unfortunately, more is not necessarily better or easier. To properly select evaluation targets for a research study the researcher must be highly aware of the features of the projects that may influence the results. Our preliminary investigation of 55 published papers indicates that this process is frequently undocumented or haphazard.

To help with this issue we developed RepoGrams, a tool for analyzing and comparing software repositories across multiple dimensions. The key idea is a flexible repository footprint abstraction that can compactly represent a variety of user-defined metrics to help characterize software projects over time. We evaluated RepoGrams in two user studies and found that it helps researchers to answer advanced, open-ended, questions about the relative evolution of software projects. RepoGrams is released as free software [53] and is made available online at <http://repograms.net/>.

# Bibliography

- [1] AngularJS — Superheroic JavaScript MVW Framework. <https://angularjs.org/>. → pages 28
- [2] CherryPy — A Minimalist Python Web Framework. <http://cherrypy.org/>. → pages 28
- [3] Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>. → pages 28
- [4] Welcome to Apache Maven. <http://maven.apache.org/>. → pages 41
- [5] Flickr uploading tool for GNOME. <https://github.com/GNOME/poslr>. → pages 4
- [6] Welcome to pygit2s documentation. <http://www.pygit2.org/>. → pages 28
- [7] DB Browser for SQLite project. <https://github.com/sqlitebrowser/sqlitebrowser>. → pages 4
- [8] Summarizing Software Artifacts. <https://www.cs.ubc.ca/cs-research/software-practices-lab/projects/summarizing-software-artifacts>. → pages 8
- [9] Alexa. github.com Site Overview. <http://www.alexa.com/siteinfo/github.com>. [Accessed Apr. 20, 2015]. → pages 1
- [10] A. Alipour, A. Hindle, and E. Stroulia. A Contextual Approach Towards More Accurate Duplicate Bug Report Detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*, pages 183–192, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487123>. → pages 66, 67

- [11] J. B. Begole, J. C. Tang, R. B. Smith, and N. Yankelovich. Work Rhythms: Analyzing Visualizations of Awareness Histories of Distributed Groups. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work, CSCW '02*, pages 334–343, New York, NY, USA, 2002. ACM. ISBN 1-58113-560-2. doi:10.1145/587078.587125. URL <http://doi.acm.org/10.1145/587078.587125>. → pages 11
- [12] N. Bettenburg, M. Nagappan, and A. E. Hassan. Think Locally, Act Globally: Improving Defect and Effort Prediction Models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, MSR '12*, pages 60–69, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1761-0. URL <http://dl.acm.org/citation.cfm?id=2664446.2664455>. → pages 64
- [13] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. The Promises and Perils of Mining Git. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR '09*, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3493-0. doi:10.1109/MSR.2009.5069475. URL <http://dx.doi.org/10.1109/MSR.2009.5069475>. → pages 5, 31, 50
- [14] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't Touch My Code! Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11*, pages 4–14, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0443-6. doi:10.1145/2025113.2025119. URL <http://doi.acm.org/10.1145/2025113.2025119>. → pages 42
- [15] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère. Orion: A Software Project Search Engine with Integrated Diverse Software Artifacts. In *Proceedings of the 2013 18th International Conference on Engineering of Complex Computer Systems, ICECCS '13*, pages 242–245, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-5007-7. doi:10.1109/ICECCS.2013.42. URL <http://dx.doi.org/10.1109/ICECCS.2013.42>. → pages 8
- [16] A. Borges, W. Ferreira, E. Barreiros, A. Almeida, L. Fonseca, E. Teixeira, D. Silva, A. Alencar, and S. Soares. Support Mechanisms to Conduct Empirical Studies in Software Engineering: A Systematic Mapping Study. In *Proceedings of the 19th International Conference on Evaluation and*

*Assessment in Software Engineering*, EASE '15, pages 22:1–22:14, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3350-4.  
doi:10.1145/2745802.2745823. URL  
<http://doi.acm.org/10.1145/2745802.2745823>. → pages 9

- [17] K. Chen, P. Liu, and Y. Zhang. Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 175–186, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi:10.1145/2568225.2568286. URL  
<http://doi.acm.org/10.1145/2568225.2568286>. → pages 1
- [18] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A System for Graph-based Visualization of the Evolution of Software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, pages 77–ff, New York, NY, USA, 2003. ACM. ISBN 1-58113-642-0.  
doi:10.1145/774833.774844. URL  
<http://doi.acm.org/10.1145/774833.774844>. → pages 10
- [19] M. D'Ambros, M. Lanza, and H. Gall. Fractal Figures: Visualizing Development Effort for CVS Entities. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '05, pages 16–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9540-9.  
doi:10.1109/VISSOF.2005.1684303. URL  
<http://dx.doi.org/10.1109/VISSOF.2005.1684303>. → pages 11
- [20] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger. Analysing Software Repositories to Understand Software Evolution. In *Software Evolution*, pages 37–67. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-76439-7.  
doi:10.1007/978-3-540-76440-3.3. URL  
<http://dx.doi.org/10.1007/978-3-540-76440-3.3>. → pages 10
- [21] R. M. de Mello, P. C. da Silva, P. Runeson, and G. H. Travassos. Towards a Framework to Support Large Scale Sampling in Software Engineering Surveys. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 48:1–48:4, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2774-9.  
doi:10.1145/2652524.2652567. URL  
<http://doi.acm.org/10.1145/2652524.2652567>. → pages 9

- [22] A. Delater and B. Paech. Tracing Requirements and Source Code during Software Development: An Empirical Study. In *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, pages 25–34. IEEE, Oct 2013. doi:10.1109/ESEM.2013.16. → pages 18
- [23] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2010. ISBN 3642079857, 9783642079856. → pages 10
- [24] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A Language and Infrastructure for Analyzing Ultra-large-scale Software Repositories. In *Proceedings of the 2013 International Conference on Software Engineering, ICSE '13*, pages 422–431, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL <http://dl.acm.org/citation.cfm?id=2486788.2486844>. → pages 8
- [25] S. G. Eick, J. L. Steffen, and E. E. Sumner, Jr. Seesoft-A Tool for Visualizing Line Oriented Software Statistics. *IEEE Trans. Softw. Eng.*, 18 (11):957–968, Nov. 1992. ISSN 0098-5589. doi:10.1109/32.177365. URL <http://dx.doi.org/10.1109/32.177365>. → pages 11
- [26] Free Software Foundation. GNU General Public License, Version 3. <https://www.gnu.org/copyleft/gpl.html>. → pages 110
- [27] G. Ghezzi and H. C. Gall. Replicating Mining Studies with SOFAS. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*, pages 363–372, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487152>. → pages 9
- [28] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. How Developers Drive Software Evolution. In *Proceedings of the Eighth International Workshop on Principles of Software Evolution, IWPSE '05*, pages 113–122, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2349-8. doi:10.1109/IWPSE.2005.21. URL <http://dx.doi.org/10.1109/IWPSE.2005.21>. → pages 10
- [29] A. Gokhale, V. Ganapathy, and Y. Padmanaban. Inferring Likely Mappings Between APIs. In *Proceedings of the 2013 International Conference on Software Engineering, ICSE '13*, pages 82–91, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL <http://dl.acm.org/citation.cfm?id=2486788.2486800>. → pages 66

- [30] G. Gousios. The GHTorrent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487132>. → pages 8, 20
- [31] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman. Lean GHTorrent: GitHub Data on Demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 384–387, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi:10.1145/2597073.2597126. URL <http://doi.acm.org/10.1145/2597073.2597126>. → pages 8, 49
- [32] V. T. Heikkilä, M. Paasivaara, and C. Lassenius. Scrumbut, but does it matter? A mixed-method study of the planning process of a multi-team scrum organization. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 85–94. IEEE, 2013. → pages 16
- [33] H. Hemmati, S. Nadi, O. Baysal, O. Kononenko, W. Wang, R. Holmes, and M. W. Godfrey. The MSR Cookbook: Mining a Decade of Research. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 343–352, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487150>. → pages 9
- [34] C. Iacob and R. Harrison. Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487094>. → pages 67
- [35] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE, Nov 2005. doi:10.1109/ISESE.2005.1541818. → pages 9
- [36] R. Just, D. Jalali, and M. D. Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 437–440, New York, NY, USA, 2014. ACM. ISBN



978-1-4503-2645-2. doi:10.1145/2610384.2628055. URL  
<http://doi.acm.org/10.1145/2610384.2628055>. → pages 8

- [37] T. Kwon and Z. Su. Detecting and Analyzing Insecure Component Usage. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 5:1–5:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1614-9. doi:10.1145/2393596.2393599. URL  
<http://doi.acm.org/10.1145/2393596.2393599>. → pages 2, 15
- [38] M. Lanza. The Evolution Matrix: Recovering Software Evolution Using Software Visualization Techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 37–42, New York, NY, USA, 2001. ACM. ISBN 1-58113-508-4. doi:10.1145/602461.602467. URL  
<http://doi.acm.org/10.1145/602461.602467>. → pages 10
- [39] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes. The Small Project Observatory: Visualizing Software Ecosystems. *Sci. Comput. Program.*, 75 (4):264–275, Apr. 2010. ISSN 0167-6423. doi:10.1016/j.scico.2009.09.004. URL <http://dx.doi.org/10.1016/j.scico.2009.09.004>. → pages 10
- [40] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey. AUSUM: Approach for Unsupervised Bug Report Summarization. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 11:1–11:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1614-9. doi:10.1145/2393596.2393607. URL  
<http://doi.acm.org/10.1145/2393596.2393607>. → pages 2, 18
- [41] T. Mens and S. Demeyer. Future Trends in Software Evolution Metrics. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 83–86, New York, NY, USA, 2001. ACM. ISBN 1-58113-508-4. doi:10.1145/602461.602476. URL  
<http://doi.acm.org/10.1145/602461.602476>. → pages 10
- [42] C. Metz. How GitHub Conquered Google, Microsoft, and Everyone Else. <http://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/>. → pages 1
- [43] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2014. → pages 4, 26

- [44] S. Nadi, C. Dietrich, R. Tartler, R. C. Holt, and D. Lohmann. Linux Variability Anomalies: What Causes Them and How Do They Get Fixed? In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 111–120, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487112>. → pages 66
- [45] M. Nagappan, T. Zimmermann, and C. Bird. Diversity in Software Engineering Research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 466–476, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2237-9. doi:10.1145/2491411.2491415. URL <http://doi.acm.org/10.1145/2491411.2491415>. → pages 2, 8
- [46] S. Neu. *Telling Evolutionary Stories with Complicity*. PhD thesis, Citeseer, 2011. → pages 10
- [47] R. Nokhbeh Zaeem and S. Khurshid. Test Input Generation Using Dynamic Programming. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 34:1–34:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1614-9. doi:10.1145/2393596.2393635. URL <http://doi.acm.org/10.1145/2393596.2393635>. → pages 66, 67
- [48] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing Multiple Evolution Metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 67–75, New York, NY, USA, 2005. ACM. ISBN 1-59593-073-6. doi:10.1145/1056018.1056027. URL <http://doi.acm.org/10.1145/1056018.1056027>. → pages 10
- [49] D. Posnett, P. Devanbu, and V. Filkov. MIC check: a correlation tactic for ESE data. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 22–31. IEEE Press, 2012. → pages 15, 66
- [50] T. Proebsting and A. M. Warren. Repeatability and Benefaction in Computer Systems Research. 2015. → pages 9
- [51] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing Software Artifacts: A Case Study of Bug Reports. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 505–514, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-719-6. doi:10.1145/1806799.1806872. URL <http://doi.acm.org/10.1145/1806799.1806872>. → pages 8

- [52] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3056-5. doi:10.1145/2635868.2635922. URL <http://doi.acm.org/10.1145/2635868.2635922>. → pages 1
- [53] D. Rozenberg, V. Poser, H. Becker, F. Kosmale, S. Becking, S. Grant, M. Maas, M. Jose, and I. Beschastnikh. RepoGrams. <https://github.com/RepoGrams/RepoGrams>. → pages 52, 110
- [54] F. Servant and J. A. Jones. History Slicing: Assisting Code-evolution Tasks. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 43:1–43:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1614-9. doi:10.1145/2393596.2393646. URL <http://doi.acm.org/10.1145/2393596.2393646>. → pages 10
- [55] J. Siegmund, N. Siegmund, and S. Apel. Views on internal and external validity in empirical software engineering. In *Proceedings of the 37th International Conference on Software Engineering, ICSE 2015*, 2015. → pages 9
- [56] F. Sokol, M. Finavaro Aniche, and M. Gerosa. MetricMiner: Supporting researchers in mining software repositories. In *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on*, pages 142–146, Sept 2013. doi:10.1109/SCAM.2013.6648195. → pages 8
- [57] M.-A. D. Storey, D. Čubranić, and D. M. German. On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 193–202, New York, NY, USA, 2005. ACM. ISBN 1-59593-073-6. doi:10.1145/1056018.1056045. URL <http://doi.acm.org/10.1145/1056018.1056045>. → pages 10
- [58] A. Strauss and J. Corbin. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. September 1998. → pages 12
- [59] C. M. B. Taylor and M. Munro. Revision Towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and*

*Analysis*, VISSOFT '02, pages 43–50, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1662-9. URL <http://dl.acm.org/citation.cfm?id=832270.833810>. → pages 10

- [60] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *Proceedings of the 2010 Asia Pacific Software Engineering Conference, APSEC '10*, pages 336–345, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4266-9. doi:10.1109/APSEC.2010.46. URL <http://dx.doi.org/10.1109/APSEC.2010.46>. → pages 8
- [61] C. Treude and M.-A. Storey. Work Item Tagging: Communicating Concerns in Collaborative Software Development. *IEEE Trans. Softw. Eng.*, 38(1): 19–34, Jan. 2012. ISSN 0098-5589. doi:10.1109/TSE.2010.91. URL <http://dx.doi.org/10.1109/TSE.2010.91>. → pages 11
- [62] J. Tsay, L. Dabbish, and J. Herbsleb. Let’s Talk About It: Evaluating Contributions Through Discussion in GitHub. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, pages 144–154, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3056-5. doi:10.1145/2635868.2635882. URL <http://doi.acm.org/10.1145/2635868.2635882>. → pages 16
- [63] F. B. Viégas, M. Wattenberg, and K. Dave. Studying Cooperation and Conflict Between Authors with History Flow Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '04*, pages 575–582, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi:10.1145/985692.985765. URL <http://doi.acm.org/10.1145/985692.985765>. → pages 11
- [64] J. Warner. Top 100 Most Popular Languages on Github. <https://jaxbot.me/articles/github-most-popular-languages>, July 2014. → pages 39
- [65] M. Wattenberg, F. B. Viégas, and K. Hollenbach. Visualizing Activity on Wikipedia with Chromograms. In *Proceedings of the 11th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part II, INTERACT'07*, pages 272–287. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 3-540-74799-0, 978-3-540-74799-4. URL <http://dl.acm.org/citation.cfm?id=1778331.1778361>. → pages 11

- [66] J. Wu, R. C. Holt, and A. E. Hassan. Exploring Software Evolution Using Spectrographs. In *Proceedings of the 11th Working Conference on Reverse Engineering, WCRE '04*, pages 80–89, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2243-2. URL <http://dl.acm.org/citation.cfm?id=1038267.1039040>. → pages 10
- [67] S. Xie, F. Khomh, and Y. Zou. An Empirical Study of the Fault-proneness of Clone Mutation and Clone Migration. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*, pages 149–158, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL <http://dl.acm.org/citation.cfm?id=2487085.2487118>. → pages 16, 18, 66
- [68] J. Yang and L. Tan. Inferring semantically related words from software context. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 161–170. IEEE Press, 2012. → pages 15

# Appendix A

## Literature survey

This appendix contains meta-data and raw results for the literature survey described in Chapter 3.

### A.1 Full protocol

This is version 6 of the protocol, which we evolved during the coding process.

#### A.1.1 Scope

Our study considers a paper to be in scope if it describe *evaluation targets* that match our definition.

#### A.1.2 Overview

1. Categorize each assigned paper along 5 dimensions. Along the way, you may need to
2. expand the codebook to accommodate previously unobserved cases. The 5 dimensions:
  - (a) code: selection criteria
  - (b) code: projects visibility
  - (c) yes/no: does the paper analyzes some feature of the projects over time
  - (d) keywords: data used in the evaluation

(e) number: number of evaluation targets

### A.1.3 Procedure

Read the abstract. Usually the abstract mentions whether the paper evaluates a tool (on rare occasions it will not be mentioned in the abstract but will be in the introduction or conclusion).

Scan the paper to find the section describing the evaluation (usually titled Evaluation or Methodology, but can have another name). Once you found the name(s) of the project(s)<sup>1</sup> that are being evaluated, search for all mentions of those names and look for a paragraph that explains the reasons for selecting those projects. Usually it will contain the key phrases “*we selected X because Y*” or “*our reasons for selecting X are Y*”.

Familiarize yourself with all the codes, apply the one that matches best. Some papers can have two or more *selection criteria* codes or *project visibility* codes apply to them. Reasons for that might be:

- The selection criteria is ambiguous
- There are two sets of projects (e.g., creating a cross-project prediction model for software defects)
- It is clear that the selection process had all of the codes apply
- Example: “*All datasets used in our case study have been obtained from the PROMISE repository, and have been reported to be as diverse of datasets as can be found in this database*” [12] — **REF** and **DIV**

Some papers clearly do not evaluate software. e.g., papers that review or critique previous papers, papers that only conduct a series of interviews, etc. In this case apply **IRR** for selection criteria only.

Some papers have a detailed explanation on their selection criteria in the *threats to validity* section. Make sure to read this section as well.

---

<sup>1</sup>Some papers do not mention the project by name (e.g., **IND** paper that does not reveal the industrial partner) in which case they would usually give the project a pseudonym or call it “our case study”, “the studied program”, etc.

## A.2 Categories

The following codes apply solely to the main evaluation(s) of the paper. They do not include preliminary works.

- **Selection criteria codes** are listed in Table 3.2.

### *Disambiguation*

- **DEV** requires that the selected project(s) have a specific development process, either followed by developers or related to some automated tool. The development process is mentioned explicitly as a one of the reasons that this project was chosen or as a requirement for the tool to operate. This does not necessarily have to be a unique feature. It could be something common, such as the existence of certain data sets, usage of various aid tools that relate to each other such as an issue tracker that integrates with version control, etc.
- **QUA** and **MET** differ in strictness. They both require that the selection criteria is somewhat indexable: codebase size, age, programming languages, team composition, popularity, program domain, etc. The difference is that **QUA** is not well-defined, there is no “function” that, given a project, returns a yes/no answer to whether or not this project fits the selection criteria. **MET** is more deterministic — either a project fits the criteria, or it does not.
- **DIV** is blurry — it may be difficult to tell if the authors are characterizing the projects they selected or if they used diversity as a criteria during project selection. Therefore, consider whether diversity is mentioned in the vicinity of selection methodology and whether it is likely that it was a selection criteria.
- **ACC** is not always added when an industrial project is studied. When there is no clear reason, other than the fact that they had access, **ACC** code should not be used. If an equivalent analysis was applied to an industrial and an open source project then **ACC** should not be used. Note that the **IND** visibility code (see Table 3.3) can still apply, even if **ACC** is not used.



- **Project visibility codes** are listed in Table 3.3.
- **Analyzes some features of the project over time (evolution)**
  - **Yes:** some aspect of the analysis studies a feature of the projects’ over time. e.g., comparing two or more releases, reviewing commit logs, inspecting bug types over time.  
Example: [49] — this paper uses the projects’ commit logs and bug history in its evaluation.
  - **No:** all aspects of the analysis make use of a single snapshot of each project. e.g., running a tool on one version of the project’s source code, comparing bug types across projects but not across time.  
Example: [47] — this paper uses the projects’ source code from a single snapshot in its evaluation.

- **Evaluated artifacts keywords**

Write in keywords that describe the evaluation targets’ artifacts from used in the paper. Examples:

- [29]: “runtime traces”
- [44]: “patches”
- [67]: “code clones”
- [10]: “bug reports”

- **What is a valid evaluation target**

A software project. For example, a codebase that evolves over time with multiple collaborators. Not, for example, an abstract model or an algorithm.

- **Number of evaluation targets**

The number of evaluation targets that the paper uses. This is a subjective number, as some targets can be thought of as 1 project or many projects (e.g., Android is an operating system with many sub-projects: One paper can evaluate Android as a single target, while another paper can evaluate the many sub-projects in Android).

Use the following rules of thumb, which are ordered in decreasing precedence (initial ones take precedence):

1. If a number is explicitly mentioned, use that number.

Example of **161** targets: “*Out of the 169 apps randomly selected, 8 apps had no reviews assigned to them which left us with 161 reviewed apps*” [34]

2. For multi-project targets, look for whether a multi-project is evaluated as a single target or as multiple targets.

Example of **8** targets: [47] — this paper names 3 targets (“Microbenchmarks”, “Google Chrome”, and “Apple Safari”), but in various tables and in the text the Microbenchmarks are being evaluated as 6 discrete targets. The total number is therefore 8: 6 microbenchmarks + 2 named applications.

3. If a number is not explicitly mentioned but the authors list names of projects and treat each project as a single target in their evaluation, count the names.

Example of **1** target: “We evaluate our approach on a large bug-report data-set from the Android project, which is a Linux-based operating system with several sub-projects” [10]

### A.2.1 Notes

Multiple selection codes may indicate a number of scenarios. For example, a paper might have selected two sets of projects independently (e.g., **ACC** for industrial Microsoft projects and **REF** for open source projects based on prior work). The two selection codes may also indicate a kind of filtering (e.g., **REF** for selecting benchmarks from prior work and **QUA** to filter these benchmarks down to a subset used in the paper).

## A.3 Raw results

Here we list the raw results from the literature survey.

**Table A.1:** Results on the initial set of 59 papers used to seed the codebook.

Title	Selection code [1]	# of evaluation targets
<b>MSR 2014</b>		
Mining energy-greedy API usage patterns in Android apps: an empirical study	UNK	55
GreenMiner: a hardware based mining software repositories software energy consumption framework	SPE	1
Mining questions about software energy consumption	IRR	
Prediction and ranking of co-change candidates for clones	QUA	6
Incremental origin analysis of source code files	QUA	7
Oops! where did that code snippet come from?	SPE	1
Works for me! characterizing non-reproducible bug reports	QUA	6
Characterizing and predicting blocking bugs in open source projects	QUA	6
An empirical study of dormant bugs	SPE	20
The promises and perils of mining GitHub	IRR	
Mining StackOverflow to turn the IDE into a self-confident programming prompter	CON	2
Mining questions asked by web developers	IRR	
Process mining multiple repositories for software defect resolution from control and organizational perspective	SPE	1
MUX: algorithm selection for software model checkers	REF	79
Improving the effectiveness of test suite through mining historical data	IND	1
Finding patterns in static analysis alerts: improving actionable alert ranking	QUA	3
Impact analysis of change requests on source code based on interaction and commit histories	QUA	1
An empirical study of just-in-time defect prediction using cross-project models	REF, QUA	11
Towards building a universal defect prediction model	POP, REF	1403
The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects	MET	3
Modern code reviews in open-source projects: which problems do they fix	QUA	2
Thesaurus-based automatic query expansion for interface-driven code search	REF	100
Estimating development effort in Free/Open source software projects by mining software repositories: a case study of OpenStack	QUA	1
An industrial case study of automatically identifying performance regression-causes	IND, REF	2
Revisiting Android reuse studies in the context of code obfuscation and library usages	POP	24379
Syntax errors just aren't natural: improving error reporting with language models	UNK	3
Do developers feel emotions? an exploratory analysis of emotions in software artifacts	REF	117
How does a typical tutorial for mobile development look like?	IRR	
Unsupervised discovery of intentional process models from event logs	SPE	1
<b>ICSE2014</b>		
Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?	IRR	
Analyze this! 145 questions for data scientists in software engineering	IRR	
The dimensions of software engineering success	IRR	
How do professionals perceive legacy systems and software modernization?	IRR	
SimRT: an automated framework to support regression testing for data races	QUA	5
Performance regression testing target prioritization via performance risk analysis	QUA	3
Code coverage for suite evaluation by developers	MET	1254
Time pressure: a controlled experiment of test case development and requirements review	IRR	
Verifying component and connector models against crosscutting structural views	UNK	4
TradeMaker: automated dynamic analysis of synthesized tradespaces	REF, CON	4
Lifting model transformations to product lines	IRR	
Automated goal operationalisation based on interpolation and SAT solving	IRR	
Mining configuration constraints: static analyses and empirical results	QUA	4
Which configuration option should I change?	QUA	8
Detecting differences across multiple instances of code clones	UNK	3
Achieving accuracy and scalability simultaneously in detecting application clones on Android markets	POP	150145
Two's company, three's a crowd: a case study of crowdsourcing software development	DES, IND	1
Does latitude hurt while longitude kills? geographical and temporal separation in a large scale software development project	DES, IND	1
Software engineering at the speed of light: how developers stay current using twitter	IRR	
Building it together: synchronous development in OSS	REF, QUA, SPE	31
A critical review of "automatic patch generation learned from human-written patches": essay on the problem statement and the evaluation of automatic software repair	IRR	
Data-guided repair of selection statements	QUA	7
The strength of random search on automated program repair	REF	7
MintHint: automated synthesis of repair hints	MET	3
Mining behavior models from user-intensive web applications	IND	1
Reviser: efficiently updating IDE-IFDS-based data-flow analyses in response to incremental program changes	DES, UNK	4
Automated design of self-adaptive software with control-theoretical formal guarantees	QUA	3
Perturbation analysis of stochastic systems with empirical distribution parameters	IRR	
How do centralized and distributed version control systems impact software changes?	MET	132
Transition from centralized to decentralized version control systems: a case study on reasons, barriers, and outcomes	IRR	

- **SPE** (“SPEcial development process required”) was renamed to **DEV** (“some quality of the DEVelopment practice required”)

- **CON** and **IND**, which were originally “selection process” codes, were used to create the “project visibility” category
- **POP** (“A complete set or random subset of projects from an explicit population of repositories (such as GitHub, an app store, etc.)”) and **MET** (“random or manual selection based on a set of well-defined METrics”) was removed from the final version of the codebook as no papers in the main literature survey were categorized using these codes. An extended literature survey might reveal such papers, in which case these codes can be re-added to the codebook.
- **DES** (“Evaluated on a project that the tool is designed for, or a case study performed on specific projects (no tool)”) was removed and replaced by other rationales where appropriate

**Table A.2: Results and analysis of the survey of 55 paper.**

Title	Selection code	Visibility code	Analyzes evolution	Evaluated data type keywords	# of evaluation t
<b>ICSE2013</b>					
Robust reconfigurations of component assemblies	IRR				
Coupling software architecture and human architecture for collaboration-aware s	IRR				
Inferring likely mappings between APIs	QUA	PUB	No	runtime traces	21
Creating a shared understanding of testing culture on a social coding site	IRR				
Human performance regression testing	QUA	PUB	No	User performance times	1
Teaching and learning programming and software engineering via interactive ga	IRR				
UML in practice	IRR				
Agility at scale: economic governance, measured improvement, and disciplined	IRR				
Reducing human effort and improving quality in peer code reviews using automa	ACC,DEV	IND	No	review requests, commits	2 or 3
Improving feature location practice with multi-faceted interactive exploration	REF,QUA	PUB	No	source code, features	1
<b>MSR2013</b>					
Which work-item updates need your response?	DEV	PUB,IND	Yes	work items	2
Linux variability anomalies: what causes them and how do they get fixed?	DEV	PUB	Yes	Patches	1
An empirical study of the fault-proneness of clone mutation and clone migration	QUA,DIV	PUB	Yes	code clones	3
A contextual approach towards more accurate duplicate bug report detection	DEV,QUA	PUB	Yes	bug reports	1
Why so complicated? simple term filtering and weighting for location-based bug	DEV,QUA	PUB	No	bug reports, source code, commits	2
The impact of tangled code changes	DEV,QUA	PUB	No	bug reports, commits	5
Replicating mining studies with SOFAS	IRR				
Bug report assignee recommendation using activity profiles	QUA,DIV	PUB	No	bug reports	3
Bug resolution catalysts: identifying essential non-committers from bug repositori	DEV,DIV	PUB,IND	No	bug reports, commits	16
Discovering, reporting, and fixing performance bugs	DEV,REF	PUB	No	bug reports, patches	3
<b>FSE2014</b>					
Verifying CTL-live properties of infinite state models using an SMT solver	IRR				
Let's talk about it: evaluating contributions through discussion in GitHub	REF,MET	PUB	Yes	pull requests, comments	?
Detecting energy bugs and hotspots in mobile apps	DIV	PUB	No	executables	30
Selection and presentation practices for code example summarization	DEV	PUB	No	code fragments	1
Vector abstraction and concretization for scalable detection of refactorings	REF,QUA	PUB	Yes	source code, commits	203
Focus-shifting patterns of OSS developers and their congruence with call graphs	QUA	PUB	Yes	commits	15
Building call graphs for embedded client-side code in dynamic web applications	REF	PUB	No	source code	5
JSAI: a static analysis platform for JavaScript	REF,DIV	PUB	No	source code	28
Sherlock: scalable deadlock detection for concurrent programs	REF,DIV	PUB	No	source code	22
Sketches and diagrams in practice	IRR				
<b>FSE2012</b>					
Detecting and analyzing insecure component usage	QUA	PUB	No	components, security policies	6
Do crosscutting concerns cause modularity problems?	DEV,QUA	PUB	Yes	bug reports, patches, reviews	1
AUSUM: approach for unsupervised bug report summarization	REF,UNK	PUB,IND	No	bug reports	2
Test input generation using dynamic programming	REF,QUA	PUB	No	source code	8
Mining the execution history of a software system to infer the best time for its ad	UNK	UNK	No	event log	1
CarFast: achieving higher statement coverage faster	IRR				
Multi-layered approach for recovering links between bug reports and fixes	REF	PUB	Yes	bug reports, commits	3
Understanding myths and realities of test-suite evolution	DEV,QUA	PUB	Yes	test suites	6
Searching connected API subgraph via text phrases	IRR				
Rubicon: bounded verification of web applications	DEV,QUA,DIV	PUB	No	source code, specs	5
<b>MSR2012</b>					
MIC check: A correlation tactic for ESE data	DEV	PUB	Yes	bug reports, commit logs	4
Think locally, act globally: Improving defect and effort prediction models	REF,DIV	PUB	Yes	defects	4
Analysis of customer satisfaction survey data	IRR				
Inferring semantically related words from software context	REF	PUB	No	source code	7
A qualitative study on performance bugs	DEV,QUA	PUB	No	bug reports	2
<b>ASE2013</b>					
Improving efficiency of dynamic analysis with dynamic dependence summaries	REF	PUB	Yes	source code	6
BitA: Coverage-guided, automatic testing of actor programs	REF,QUA	PUB,CON	No	source code	8
Ranger: Parallel analysis of alloy models by range partitioning	IRR				
JFlow: Practical refactorings for flow-based parallelism	REF,DIV	PUB	No	source code	7
SEDGE: Symbolic example data generation for dataflow programs	REF,QUA	PUB	No	source code	31
<b>ESEM2013</b>					
Tracing Requirements and Source Code during Software Development: An Emp	DEV	CON	Yes	requirements, work items, source cod	3
When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing	REF,DEV	PUB	Yes	commits, source code, vulnerabilities	1
ScrumBut, But Does it Matter? A Mixed-Method Study of the Planning Process o	DEV,ACC	IND	Yes	requirements	1
Using Ensembles for Web Effort Estimation	IRR				
Experimental Comparison of Two Safety Analysis Methods and Its Replication	IRR				


## Appendix B

# Undergraduate students study

This appendix contains meta-data and raw results for the user study with undergraduate students described in Section 5.1.

### B.1 Slides from the in-class demonstration


---

The logo for 'Repo grams' features the word 'Repo' in a bold, black, sans-serif font above the word 'grams' in a similar font. The text is set against a background of horizontal, multi-colored stripes in shades of purple, blue, green, and yellow.


Brief lecture and  
in-class research study

A tool to analyze and juxtapose  
software project history

---

The logo of the University of British Columbia (UBC), featuring a shield with a blue top section containing the letters 'UBC', a yellow sunburst at the bottom, and blue wavy lines representing water in the middle.

Software Practices Lab  
Computer Science  
University of British Columbia

The logo of Saarland University, depicting a stylized blue owl with large, circular eyes and a body composed of vertical bars.

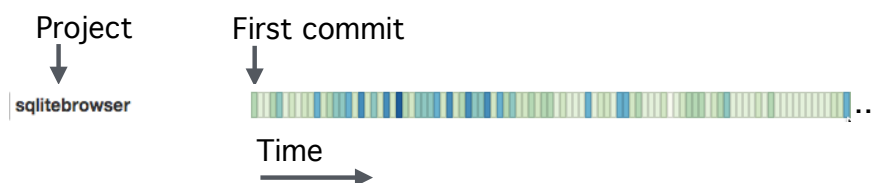
Computer Science  
Saarland University

## Another tool to visualize repositories?

- There are numerous tool to visualize repositories
- None provide a **flexible** interface to
  - Juxtapose/compare multiple repositories
  - Unify multiple metrics of a repository into one view
  - Simple and easy to use
- Our targeted population: **SE researchers**
  - Need to select evaluation targets for studies
  - Need a simple and efficient project comparison tool

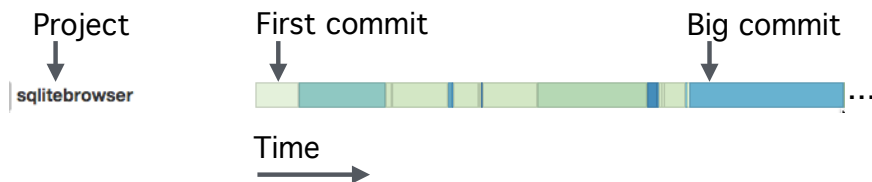
## Repograms: a repository is a **sequence of blocks**

- A **block** represents a **commit**
  - Block's length is either a fixed constant or encodes lines of codes changed
  - A block's colour represents a "metric" value
  - A metric is a function:  $m(commit) \rightarrow number$
- Example (**block length = fixed constant**):



## Reprograms: a repository is a sequence of blocks

- A **block** represents a **commit**
  - Block's length is either a fixed constant or encodes lines of codes changed
  - A block's colour represents a "metric" value
  - A metric is a function:  $m(commit) \rightarrow number$
- Example (**block length = lines of code changed**):

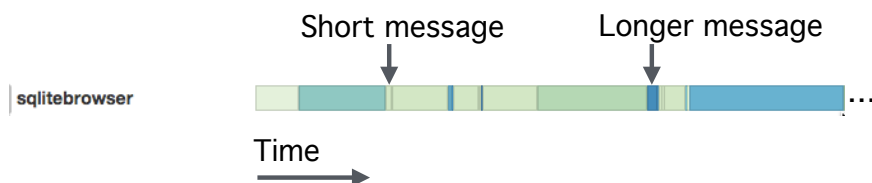
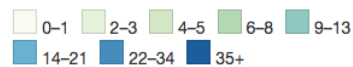


Ivan Beschastnikh

University of British Columbia 4

## Reprograms: a repository is a sequence of blocks

- A **block** represents a **commit**
  - Block's length is either fixed constant or encodes **lines of codes changed**
  - A block's colour represents a "metric" value
  - Example metric: "**number of words in a commit message**"



Ivan Beschastnikh

University of British Columbia 5





## Human subjects review (REB)

---

- REB: research **ethics** board
  - Independent body
  - Reviews research protocol + study materials
  - Goal is to maximize human safety: protect human subjects from physical or psychological harm
  - Risk-benefit analysis

## Repograms REB application

---

- PDFs

---

## In-class research study



- *You* are the subjects
- Voluntary (you don't have to participate)
- Can do the study at home (anytime this week)
- Enter raffle for 5 x \$25 gift cards to UBC Bookstore

---

Help us evaluate  !

---

Browser options: **Chrome (best)**, Firefox, or Safari (worst)

Does **not** work in IE

Avoid tablets and phones

**Begin study by browsing to:**

**<http://repograms.net>**

---

Questions? Raise your hand.

## B.2 Protocol and questionnaire

### B.2.1 Overview

The study was conducted in a 4th year software engineering class at the University of British Columbia. Prior to the study there were two lectures by the leading investigator. The first lecture covered concepts in version control systems and research methods. The second lecture was an introduction to RepoGrams. The slides from the latter lecture are presented in the preceding section.

Participation was voluntary, we emphasized this before beginning the study. There were 105 students in the classroom, 91 students began the questionnaire and 74 completed it.

When opening RepoGrams during this study it automatically loaded the following 5 repositories<sup>1</sup>:

- <https://github.com/RepoGrams/sqlitebrowser>
- <https://github.com/RepoGrams/vim.js>
- <https://github.com/RepoGrams/AudioStreamer>
- <https://github.com/RepoGrams/LightTable>
- <https://github.com/RepoGrams/html-pipeline>

### B.2.2 Questionnaire

[Questions are elaborated in the next sections]

We mark our ground truth answers with an underline. For some questions there is more than one correct answer.

A. Consent form

B. Demographics (1 page, 5 questions)

---

<sup>1</sup>In the study we loaded the original repositories from which these repositories were forked. We forked and froze these repositories post-study for reproducibility reasons.

- C. Warmup questions (2 pages, 4 questions)
- D. Metric comprehension questions (5 pages, 6 questions)
- E. Questions about comparisons across projects (3 pages, 3 questions)
- F. Exploratory question (1 page, 1 question)
- G. Open comments (1 page, 2 questions)
- H. Raffle ticket

### **B.2.3 Demographics**

- a. How many Computer Science courses have you successfully passed?
  - 0–5 courses
  - 6–10 courses
  - 11–15 courses
  - 16 or more courses
- b. Have you worked in Computer Science related jobs (e.g. co-ops, internships)? If so, how many academic terms have you spent working in such jobs?
  - I have not worked in CS related jobs
  - 1 term
  - 2 terms
  - 3 terms
  - 4 or more terms
- c. How many years of experience have you had with version control systems? (e.g., mercurial, git, svn, cvs)
  - No experience

- 1 year of experience
- 2 years of experience
- 3 years of experience
- 4 or more years of experience

d. How often do you use version control systems?

Consider any use of version control systems, whether it is during courses, jobs, or for private use.

- Daily
- Weekly
- Monthly
- Never use/used

e. Have you ever used version control outside of class, for a project not related to schoolwork?

- Yes
- No

f. Note: Our online questionnaire included another question in the Demographics section after e., however, we found that the question was understood ambiguously by a large number of participants. As a consequence we removed that question from our analysis and do not report on it here.

#### **B.2.4 Warmup questions**

(pg. 1)

- We developed a tool called RepoGrams for comparing and analyzing the history of multiple software projects. In this study you will help us evaluate RepoGrams.

- RepoGrams can be used to analyze various repository metrics, such as the number of programming languages that a project consists of over time.
- A repository consists of a number of commits. RepoGrams represents a commit as a block. The blocks can be configured to have different block lengths.
- A metric measures something about a commit. RepoGrams calculates a numeric metric value for each commit in a project repository.
- RepoGrams uses colours to represent different commit metric values. The legend details which colour corresponds to each metric value, or range of values.

Open RepoGrams in a separate tab: <http://reporgrams.net:2000/>

In RepoGrams, a row of blocks represents the software repository history of a software project. A coloured block in one such row represents a single commit to a project repository.

1. How many projects are listed in the newly opened RepoGrams tab?
  - 1
  - 2
  - 3
  - 4
  - 5
2. Using the Fixed block length, estimate the number of commits in the **AudioStreamer** project using the RepoGrams tool:
  - Tens (10–99)
  - Hundreds (100–999)
  - Thousands (1,000–9,999)
  - Tens of thousands (10,000–99,999)

(pg. 2)

*How to use RepoGrams*

- Hover your mouse pointer over the block that represents the commit (desktop/laptop) or touch the commit (tablets) to see the commit's unique identifier (called commit hash) and the commit message. Clicking/touching the commit will copy the first 5 letters/numbers of the commit's hash to your clipboard for easy pasting into the answer sheet.
  - When asked to identify one or more commits, type the first 5 letters/numbers of their commit hash to the answer sheet.
  - When asked to identify one or more projects, project repositories, or metric, type their name(s) to the answer sheet.
3. Remove the **AudioStreamer** project repository from the view. How many project are now listed?

- 1
- 2
- 3
- 4
- 5

4. Add the **Postr** repository (URL provided below) to the view. Using any metric and the Lines changed (comparable btw. projects) block length and the zoom control, how does the **Postr** project compare with the other projects in terms of its size? (Options below are listed from smallest to largest)

<https://github.com/RepoGrams/postr>

- Postr is the smallest project
- Postr is the second smallest project
- Postr is the second largest project
- Postr is the largest project



## B.2.5 Metric comprehension questions

(pg. 1)

### *Languages in a Commit metric*

The Languages in a Commit metric measures the number of different programming languages used in each commit. For example, a commit that changed one Java file and two XML files would get the value 2 because it changed a Java file and XML files. A commit that changed 100 Java files would get the value 1 because it only changed Java files.

5. Using the Languages in a Commit metric and any block length, which project is likely to contain source code written in the most diverse number of different languages?
  - `sqlitebrowser`
  - `vim.js`
  - `LightTable`
  - `html-pipeline`
  - `postr`

(pg. 2)

### *Branches Used metric*

The Branches Used metric assigns a colour to each commit based on the branch that the commit belongs to (each branch in a project is given a unique colour).

6. Using the Branches Used metric and the Lines changed (incomparable btw. projects) block length, which project repository is most like to have the **least** number of distinct branches?
  - `sqlitebrowser`
  - `vim.js`
  - `LightTable`
  - `html-pipeline`

- postr

7. Using the Branches Used metric and the Lines changed (incomparable btw. projects) block length, which project repository is most like to have the **most** number of distinct branches?

- sqlitebrowser
- vim.js
- LightTable
- html-pipeline
- postr

(pg. 3)

*Most Edited File metric*

The Most Edited Files metric measures the number of times that the most edited file in a commit has been previously modified. A commit with a high metric value indicates that it modifies a file that was changed many times in previous commits. A commit will have a low value if it is composed of new or rarely edited files.

8. Using the Most Edited File metric and the Fixed block length, what is the commit hash of the latest commit that modified the most popular file(s) in the Postr project?

*[text field]*

[ground truth answer: bed3257]

(pg. 4)

*Commit Localization metric*

The Commit Localization metric represents the fraction of the number of unique project directories containing files modified by the commit.

Metric value of 1 means that all the modified files in a commit are in a single directory. Metric value of 0 means all the project directories contain a file modified by the commit.

9. Using the Commit Localization metric and the Fixed block length, which project had the longest uninterrupted sequence of commits with metric values in the range 0.88–1.00?

- [sqlitebrowser](#)
- vim.js
- LightTable
- html-pipeline
- postr

(pg. 5)

#### *Number of Branches metric*

The Number of Branches metric measures the number of branches actively used by developers: a high value means that developers were making changes in many different branches at the time that a commit was created.

10. Using the Number of Branches metric and the Fixed block length, find one commit in **LightTable** when developers were using the largest number of concurrent branches?

There can be multiple correct answers to this question.

*[text field]*

[ground truth answer: any of [522d848](#), [7729887](#), [c55f44e](#)]

### **B.2.6 Questions about comparisons across projects**

In this set of questions you will be asked to compare different projects based on one or two metrics. For each of the following questions, explain the reason for your choices in 1–2 short sentences.

(pg. 1)

11. Using the Number of Branches metric and the Lines changed (incomparable btw. projects) block length, which project appears to have a development process most similar to **LightTable**?

- [sqlitebrowser](#)
- [vim.js](#)
- [LightTable](#)
- [html-pipeline](#)
- [postr](#)

a. Why did you choose this project?

*[text field]*

(pg. 2)

12. Using the Languages in a Commit metric and the Fixed block length, which two project repositories appear to have the most similar development process with each other?

- [sqlitebrowser](#)
- [vim.js](#)
- [LightTable](#)
- [html-pipeline](#)
- [postr](#)

a. Why did you choose this project?

*[text field]*

(pg. 3)

### *Commit Message Length*

The Commit Message Length metric counts the number of words in the commit log message of each commit.

13. Using the Commit Message Length metric and the Fixed block length, which project has a distinct pattern in the level of detail of its commit messages that distinguishes it from all of the other projects?

- [sqlitebrowser](#)

- [vim.js](#)
- LightTable
- html-pipeline
- postr
  - a. Read through some of the commit messages (by hovering over the commits) from the project you selected and briefly explain why this project has such a distinct pattern.  
*[text field]*

## B.2.7 Exploratory question

(pg. 1)

### *Choosing metrics*

Commit messages provide current and future developers in the project with insight for what changes were made and the reasoning behind those changes. Unfortunately, developers sometimes neglect to detail the changes and reasons.

Using the Lines changed (incomparable btw. projects) block length, find a single commit in **vim.js** where the developer made a significant change but neglected to elaborate on that change. Choose any commit that is NOT the first commit in the project. There can be multiple correct answers to this question.

14. Which metric(s) did you use?

*[text field]*

Which commit did you identify?

*[text field]*

[ground truth answer: any of [1557ac8](#), [09039d7](#), [c343fb6](#), [e071941](#), [238dba7](#), [7390e4e](#)]

- a. Write a short explanation for your choice of metric(s):

*[text field]*

### **B.2.8 Open comments**

- a. Was there anything confusing about RepoGrams? What tasks were difficult to perform?

*[text field]*

- b. Other comments about RepoGrams and this study:

*[text field]*

### **B.2.9 Filtering results**

We discarded individual answers when students spent a disproportionately short time (<10 seconds) on the page that contained those questions. We also received one extra entry (a total of 75 responses) in which the participant answered 3 of the 4 warmup questions wrong. We discarded this entry from our analysis and from all reports. Anonymized results are collected in the next section.

## **B.3 Raw results**

Here we list the raw results. Cells with a dark red background denote an answer that is incompatible with our ground truth answers. Cells with an orange background denote that the answer for this question was discarded from the report for being answered in less than 10 seconds, and was most likely skipped by the participant. Columns titled *gt* mark whether the raw answer was compatible with our ground truth (true/false). Columns titled *gt-agree* and *gt-disagree* are the encoded labels that we assigned to the explanation by the participants, based on their raw text response. We do not report raw text responses to preserve the anonymity of the participants.

**Table B.1:** Raw results from the demographics section in the user study with undergraduate students.

P	a	b	c	d	e	time
1	16+	0	4+	Weekly	yes	00:01:17
2	11-15	0	1	Monthly	no	00:00:53
3	11-15	3	3	Monthly	no	00:01:14
4	16+	3	2	Daily	yes	00:00:11
5	0-5	0	2	Weekly	no	00:02:09
6	16+	3	1	Never	yes	00:00:46
7	16+	0	2	Weekly	yes	00:00:46
8	6-10	4+	2	Daily	yes	00:00:56
9	6-10	0	1	Daily	no	00:01:50
10	11-15	2	2	Daily	yes	00:02:15
11	11-15	0	1	Weekly	yes	00:01:43
12	6-10	2	1	Daily	yes	00:00:17
13	16+	0	1	Weekly	yes	00:00:38
14	6-10	0	3	Daily	yes	00:00:19
15	6-10	0	3	Daily	yes	00:01:36
16	0-5	0	2	Weekly	no	00:02:25
17	6-10	2	1	Weekly	yes	00:01:36
18	11-15	2	1	Daily	yes	00:01:07
19	0-5	3	3	Daily	yes	00:00:28
20	11-15	4+	4+	Weekly	yes	00:01:23
21	11-15	3	2	Monthly	yes	00:00:31
22	16+	0	2	Daily	yes	00:00:29
23	6-10	2	1	Monthly	yes	00:00:15
24	6-10	0	1	Weekly	yes	00:01:11
25	11-15	3	1	Weekly	yes	00:00:47
26	11-15	2	1	Daily	yes	00:01:02
27	11-15	0	2	Weekly	yes	00:00:55
28	11-15	0	2	Weekly	yes	00:02:25
29	6-10	2	2	Daily	yes	00:01:47
30	11-15	2	2	Monthly	yes	00:00:50
31	11-15	3	3	Weekly	yes	00:00:37

P	a	b	c	d	e	time
32	16+	0	2	Never	yes	00:00:53
33	0-5	1	2	Weekly	yes	00:01:11
34	11-15	1	1	Monthly	no	00:01:28
35	6-10	1	1	Monthly	yes	00:01:07
36	0-5	0	2	Weekly	yes	00:00:57
37	6-10	2	2	Daily	yes	00:00:30
38	11-15	4+	2	Weekly	yes	00:00:36
39	11-15	4+	2	Weekly	no	00:00:59
40	0-5	0	1	Daily	yes	00:00:44
41	11-15	3	2	Daily	yes	00:00:38
42	11-15	4+	4+	Weekly	yes	00:00:54
43	6-10	0	3	Monthly	yes	00:00:59
44	11-15	0	0	Never	no	00:00:41
45	11-15	4+	2	Weekly	yes	00:01:13
46	6-10	2	1	Weekly	no	00:01:47
47	11-15	0	2	Daily	yes	00:01:45
48	16+	4+	2	Weekly	yes	00:00:50
49	11-15	1	1	Weekly	yes	00:01:52
50	6-10	2	1	Weekly	yes	00:01:25
51	11-15	1	1	Never	yes	00:04:19
52	0-5	0	2	Monthly	no	00:01:42
53	16+	4+	3	Weekly	yes	00:02:07
54	6-10	1	2	Weekly	yes	00:01:15
55	11-15	2	1	Weekly	yes	00:01:31
56	11-15	2	2	Daily	yes	00:00:30
57	6-10	0	1	Weekly	no	00:01:29
58	16+	4+	3	Daily	yes	00:01:46
59	11-15	1	0	Never	no	00:02:10
60	6-10	2	1	Weekly	yes	00:01:10
61	6-10	2	1	Daily	yes	00:01:01
62	11-15	3	3	Daily	yes	00:00:50

P	a	b	c	d	e	time
63	6-10	0	1	Monthly	no	00:00:54
64	0-5	0	2	Daily	yes	00:01:36
65	16+	3	3	Weekly	yes	00:00:31
66	16+	4+	3	Daily	yes	00:01:01
67	6-10	0	1	Never	no	00:00:57
68	16+	4+	3	Monthly	yes	00:03:26
69	6-10	2	2	Weekly	yes	00:00:38
70	6-10	4+	1	Never	yes	00:03:56
71	16+	2	1	Daily	yes	00:00:35
72	6-10	4+	1	Monthly	no	00:01:12
73	16+	3	3	Weekly	yes	00:02:27
74	11-15	0	2	Weekly	yes	00:00:38

**Table B.2:** Raw results from the warmup section (questions 1–4) in the user study with undergraduate students.

Red cells denote participant answers that do not match our ground truth for that question.

P	1	2	time	3	4	time
1	5	Tens	00:01:55	4	Second smallest	00:00:07
2	5	Tens	00:04:20	4	Second smallest	00:03:43
3	5	Tens	00:02:44	4	Second smallest	00:02:11
4	5	Tens	00:01:37	4	Smallest	00:00:47
5	5	Tens	00:01:45	4	Smallest	00:01:06
6	5	Tens	00:02:45	4	Second smallest	00:02:05
7	5	Tens	00:02:09	4	Second smallest	00:02:05
8	5	Tens	00:01:56	4	Second smallest	00:01:35
9	5	Tens	00:02:29	4	Smallest	00:02:03
10	5	Tens	00:01:31	4	Second smallest	00:00:03
11	5	Tens	00:04:15	4	Smallest	00:01:13
12	5	Tens	00:01:09	4	Second smallest	00:01:47
13	5	Tens	00:01:50	4	Smallest	00:01:54
14	5	Tens	00:00:47	4	Largest	00:00:40
15	5	Tens	00:02:36	4	Second smallest	00:03:21
16	5	Tens	00:01:26	4	Smallest	00:00:44
17	5	Tens	00:02:13	4	Second smallest	00:01:44
18	5	Hundreds	00:02:38	4	Second smallest	00:02:31
19	5	Tens	00:01:12	4	Second largest	00:00:40
20	5	Tens	00:03:08	4	Second smallest	00:02:04
21	5	Hundreds	00:03:36	4	Second smallest	00:02:08
22	5	Tens	00:01:37	4	Second largest	00:05:15
23	5	Tens	00:00:03	4	Second smallest	00:00:10
24	5	Tens	00:02:35	4	Smallest	00:01:46
25	5	Tens	00:01:32	4	Smallest	00:01:44
26	5	Tens	00:02:17	4	Second smallest	00:01:54
27	5	Tens	00:01:27	4	Second smallest	00:01:31
28	5	Tens	00:02:45	4	Second smallest	00:01:47
29	5	Tens	00:01:56	4	Second smallest	00:01:21
30	5	Tens	00:01:10	4	Second smallest	00:01:45
31	5	Tens	00:02:07	4	Largest	00:01:17
32	5	Tens	00:04:02	4	Smallest	00:00:46
33	5	Tens	00:04:37	4	Second smallest	00:02:01
34	5	Tens	00:02:52	4	Second smallest	00:02:01
35	5	Hundreds	00:02:13	4	Second smallest	00:01:12
36	5	Tens	00:02:30	4	Smallest	00:05:17
37	5	Tens	00:06:34	4	Smallest	00:01:01
38	5	Hundreds	00:01:46	4	Second smallest	00:00:52
39	5	Hundreds	00:04:00	4	Smallest	00:03:02
40	5	Tens	00:01:31	4	Second largest	00:00:54
41	5	Tens	00:03:16	4	Second smallest	00:00:10
42	5	Tens	00:02:04	4	Smallest	00:02:31
43	5	Tens	00:02:26	4	Smallest	00:02:43
44	5	Tens	00:03:46	4	Second smallest	00:01:50
45	5	Tens	00:01:57	4	Second smallest	00:03:01
46	5	Thousands	00:06:38	4	Second largest	00:02:17
47	5	Tens	00:01:04	4	Smallest	00:00:54
48	5	Hundreds	00:04:01	4	Second smallest	00:02:11
49	5	Tens	00:01:10	4	Second smallest	00:01:27
50	5	Tens	00:02:13	4	Second smallest	00:03:51
51	5	Tens	00:02:33	4	Smallest	00:01:32
52	5	Tens	00:01:30	4	Smallest	00:01:22
53	5	Tens	00:04:09	4	Second smallest	00:01:47
54	5	Tens	00:07:07	4	Second smallest	00:00:08
55	5	Tens	00:02:48	4	Second smallest	00:00:55
56	5	Tens	00:00:58	4	Second smallest	00:00:45
57	5	Tens	00:02:13	4	Second smallest	00:02:13
58	5	Tens	00:01:56	4	Second smallest	00:00:49
59	5	Tens	00:05:41	4	Second smallest	00:01:28
60	5	Tens	00:00:53	4	Second smallest	00:02:09
61	5	Tens	00:01:40	4	Second smallest	00:03:12
62	5	Tens	00:02:46	4	Second smallest	00:01:39

P	1	2	time	3	4	time
63	5	Tens	00:02:46	4	Second smallest	00:01:13
64	5	Tens	00:04:21	4	Second smallest	00:01:48
65	5	Tens	00:02:15	4	Second smallest	00:03:09
66	5	Tens	00:01:21	4	Second smallest	00:01:04
67	5	Thousands	00:01:32	4	Largest	00:00:56
68	5	Tens	00:01:46	4	Smallest	00:03:06
69	5	Tens	00:00:19	4	Second smallest	00:01:12
70	5	Tens	00:03:57	4	Smallest	00:01:32
71	5	Tens	00:00:06	4	Second smallest	00:01:13
72	5	Tens	00:02:20	4	Second smallest	00:00:51
73	5	Tens	00:03:06	4	Second smallest	00:00:10
74	5	Hundreds	00:03:16	4	Second smallest	00:01:32



**Table B.3:** Raw results from the metrics comprehension section (questions 5–7) in the user study with undergraduate students.

Red cells denote participant answers that do not match our ground truth for that question.  
Orange cells denote participant answers that took less than 10 seconds, and were ignored.

P	5	time	6	7	time
1	vim.js	00:00:38	vim.js	html-pipeline	00:01:39
2	vim.js	00:02:44	vim.js	html-pipeline	00:01:21
3	vim.js	00:00:56	vim.js	html-pipeline	00:01:14
4	vim.js	00:00:16	vim.js	html-pipeline	00:00:55
5	LightTable	00:00:57	vim.js	html-pipeline	00:01:22
6	vim.js	00:01:29	vim.js	html-pipeline	00:01:26
7	vim.js	00:01:18	sqlitebrowser	html-pipeline	00:02:38
8	vim.js	00:01:22	vim.js	html-pipeline	00:02:04
9	vim.js	00:01:19	vim.js	html-pipeline	00:00:47
10	skipped		vim.js	html-pipeline	00:03:51
11	skipped		vim.js	html-pipeline	00:01:14
12	LightTable	00:01:38	vim.js	html-pipeline	00:01:23
13	LightTable	00:00:30	vim.js	LightTable	00:01:05
14	vim.js	00:00:53	vim.js	html-pipeline	00:00:42
15	vim.js	00:01:39	vim.js	html-pipeline	00:00:11
16	vim.js	00:00:37	vim.js	html-pipeline	00:02:13
17	vim.js	00:01:35	vim.js	html-pipeline	00:01:15
18	vim.js	00:04:35	sqlitebrowser	html-pipeline	00:01:53
19	vim.js	00:00:55	vim.js	html-pipeline	00:00:42
20	vim.js	00:00:53	vim.js	html-pipeline	00:01:53
21	vim.js	00:01:38	postr	LightTable	00:00:43
22	vim.js	00:01:44	vim.js	html-pipeline	00:01:05
23	skipped		vim.js	html-pipeline	00:00:39
24	vim.js	00:01:32	vim.js	html-pipeline	00:01:45
25	LightTable	00:01:29	sqlitebrowser	LightTable	00:01:44
26	vim.js	00:00:40	LightTable	vim.js	00:01:20
27	LightTable	00:01:27	vim.js	html-pipeline	00:00:42
28	vim.js	00:00:40	vim.js	html-pipeline	00:01:35
29	vim.js	00:00:43	vim.js	html-pipeline	00:01:20
30	vim.js	00:00:37	html-pipeline	vim.js	00:00:31
31	LightTable	00:02:24	vim.js	html-pipeline	00:00:59
32	sqlitebrowser	00:01:22	vim.js	html-pipeline	00:01:58
33	vim.js	00:07:44	vim.js	html-pipeline	00:01:20
34	vim.js	00:00:54	vim.js	html-pipeline	00:01:52
35	vim.js	00:00:29	vim.js	html-pipeline	00:01:33
36	vim.js	00:01:05	vim.js	html-pipeline	00:02:35
37	LightTable	00:00:36	vim.js	html-pipeline	00:01:46
38	LightTable	00:00:20	vim.js	html-pipeline	00:00:42
39	vim.js	00:01:30	vim.js	html-pipeline	00:01:22
40	postr	00:00:35	LightTable	html-pipeline	00:02:45
41	vim.js	00:00:37	LightTable	html-pipeline	00:01:23

P	5	time	6	7	time
42	LightTable	00:01:41	vim.js	html-pipeline	00:01:47
43	vim.js	00:02:34	sqlitebrowser	html-pipeline	00:03:10
44	vim.js	00:00:58	vim.js	html-pipeline	00:01:35
45	LightTable	00:01:38	vim.js	html-pipeline	00:07:43
46	skipped		vim.js	html-pipeline	00:02:34
47	vim.js	00:01:21	vim.js	html-pipeline	00:00:56
48	vim.js	00:01:25	vim.js	html-pipeline	00:01:09
49	vim.js	00:01:11	sqlitebrowser	html-pipeline	00:00:50
50	vim.js	00:01:21	vim.js	html-pipeline	00:02:39
51	skipped		vim.js	html-pipeline	00:01:00
52	LightTable	00:01:02	html-pipeline	vim.js	00:01:52
53	LightTable	00:03:35	vim.js	html-pipeline	00:01:12
54	skipped		skipped	skipped	
55	vim.js	00:00:50	html-pipeline	LightTable	00:00:52
56	vim.js	00:00:31	vim.js	postr	00:01:21
57	skipped		skipped	skipped	
58	vim.js	00:00:45	postr	sqlitebrowser	00:01:19
59	vim.js	00:00:55	vim.js	html-pipeline	00:01:08
60	sqlitebrowser	00:00:52	vim.js	html-pipeline	00:01:00
61	vim.js	00:00:53	vim.js	html-pipeline	00:01:56
62	vim.js	00:00:57	vim.js	html-pipeline	00:01:19
63	vim.js	00:01:39	skipped	skipped	
64	vim.js	00:01:16	vim.js	html-pipeline	00:01:21
65	LightTable	00:01:36	vim.js	html-pipeline	00:01:58
66	LightTable	00:00:41	vim.js	postr	00:01:04
67	vim.js	00:01:29	vim.js	html-pipeline	00:01:00
68	sqlitebrowser	00:01:09	vim.js	html-pipeline	00:01:06
69	vim.js	00:00:55	vim.js	html-pipeline	00:00:49
70	LightTable	00:01:33	vim.js	html-pipeline	00:01:15
71	vim.js	00:00:51	vim.js	html-pipeline	00:01:54
72	vim.js	00:00:35	sqlitebrowser	html-pipeline	00:00:51
73	vim.js	00:00:15	vim.js	html-pipeline	00:01:12
74	vim.js	00:00:38	LightTable	vim.js	00:01:04

**Table B.4:** Raw results from the metrics comprehension section (questions 8–10) in the user study with undergraduate students.

Red cells denote participant answers that do not match our ground truth for that question.  
Orange cells denote participant answers that took less than 10 seconds, and were ignored.

P	8		time		9	time		10		time
1	bed325	TRUE	00:01:04		sqlitebrowser	00:01:15		c55f44e	TRUE	00:01:04
2	bed325	TRUE	00:03:01		sqlitebrowser	00:03:42		c55f44e	TRUE	00:01:44
3	bed325	TRUE	00:01:36		sqlitebrowser	00:00:52		c55f44e	TRUE	00:01:03
4	bed325	TRUE	00:00:44		sqlitebrowser	00:00:33		80ac53	FALSE	00:00:34
5	bed325	TRUE	00:02:13		sqlitebrowser	00:01:52		c55f44e	TRUE	00:01:19
6	bed325	TRUE	00:02:20		sqlitebrowser	00:01:30		522d84	TRUE	00:01:01
7	bed325	TRUE	00:06:15		sqlitebrowser	00:02:16		c55f44e	TRUE	00:00:57
8	77b4d	FALSE	00:01:08		sqlitebrowser	00:00:53		b1560	FALSE	00:00:41
9	2f1b685	FALSE	00:01:27		sqlitebrowser	00:00:32		4dc0bb	FALSE	00:00:47
10	bed325	TRUE	00:01:14		postr	00:01:10		c55f44e	TRUE	00:01:26
11	bed325	TRUE	00:01:23		sqlitebrowser	00:01:17		f12727f	FALSE	00:00:43
12	bed325	TRUE	00:00:54		sqlitebrowser	00:00:49		c55f44e	TRUE	00:00:56
13	bed325	TRUE	00:00:43		sqlitebrowser	00:00:55		7e8e21	FALSE	00:00:53
14	b8d043	FALSE	00:00:26		LightTable	00:02:02		3dad78	FALSE	00:00:31
15	bed32	TRUE	00:01:58		sqlitebrowser	00:01:52		c55f4	TRUE	00:01:20
16	bed325	TRUE	00:02:16		sqlitebrowser	00:01:37		772988	TRUE	00:00:31
17	bed325	TRUE	00:01:23		sqlitebrowser	00:01:13		c55f44e	TRUE	00:00:49
18	68ab71	FALSE	00:00:51		sqlitebrowser	00:00:43		d22605	FALSE	00:00:40
19	bed325	TRUE	00:00:51		sqlitebrowser	00:01:01		772988	TRUE	00:00:46
20	bed325	TRUE	00:01:50		vim.js	00:00:38		522d84	TRUE	00:00:39
21	e2f651ff	FALSE	00:01:37		sqlitebrowser	00:00:46		c55f44e	TRUE	00:02:29
22	b8d043	FALSE	00:01:01		sqlitebrowser	00:01:08		21e3dd	FALSE	00:00:52
23	d3566	FALSE	00:00:24		sqlitebrowser	00:00:25		77298	TRUE	00:00:42
24					sqlitebrowser	00:02:10		772988	TRUE	00:01:00
25	bed325	TRUE	00:01:00		sqlitebrowser	00:01:23		c55f44e	TRUE	00:01:24
26	944e02	FALSE	00:01:45		html-pipeline	00:01:41		c55f44e	TRUE	00:00:49
27	bed325	TRUE	00:01:00		sqlitebrowser	00:01:38		c55f44e	TRUE	00:00:40
28	bed325	TRUE	00:01:36		sqlitebrowser	00:01:54		c55f44e	TRUE	00:02:24
29	77b4d5	FALSE	00:00:53		sqlitebrowser	00:00:53		4dc0bb	FALSE	00:00:47
30	bed325	TRUE	00:01:47		sqlitebrowser	00:01:48		c55f44e	TRUE	00:00:54
31	284a48	FALSE	00:01:15		sqlitebrowser	00:00:30		4dc0bb	FALSE	00:01:14
32	bed325	TRUE	00:01:30		postr	00:00:37		023441	FALSE	00:01:10
33	ca78b0	FALSE	00:01:15		sqlitebrowser	00:02:55		522d84	TRUE	00:02:18
34	f8a1769	FALSE	00:02:19		sqlitebrowser	00:01:47		c55f44e	TRUE	00:00:58
35	bed325	TRUE	00:01:36		sqlitebrowser	00:00:51		c55f44e	TRUE	00:00:54
36	bed325	TRUE	00:01:41		sqlitebrowser	00:03:19		c55f44e	TRUE	00:01:44
37	f829f06	FALSE	00:01:24		sqlitebrowser	00:01:10		023441	FALSE	00:01:03
38	944e02	FALSE	00:00:34		sqlitebrowser	00:00:44		772988	TRUE	00:00:39
39	bed325	TRUE	00:01:20		sqlitebrowser	00:01:26		c55f44e	TRUE	00:01:20
40	b8d043	FALSE	00:01:41		postr	00:00:53		522d84	TRUE	00:01:16
41	bed325	TRUE	00:00:42		sqlitebrowser	00:00:51		772988	TRUE	00:00:37

P	8		time		9	time		10		time
42	bed325	TRUE	00:01:24		sqlitebrowser	00:02:03				
43	bed325	TRUE	00:01:21		sqlitebrowser	00:01:43		772988	TRUE	00:01:13
44	bc159	FALSE	00:02:15		sqlitebrowser	00:01:15		4dc0b	FALSE	00:01:47
45	944e02	FALSE	00:04:23		sqlitebrowser	00:05:12		772988	TRUE	00:02:13
46	944e02	FALSE	00:02:39		sqlitebrowser	00:01:01				
47	bed325	TRUE	00:01:37		sqlitebrowser	00:01:14		772988	TRUE	00:00:32
48	bed325	TRUE	00:01:30		sqlitebrowser	00:03:00		c55f44e	TRUE	00:00:51
49	944e02	FALSE	00:00:46		sqlitebrowser	00:00:38		772988	TRUE	00:00:42
50	bed32	TRUE	00:06:00		sqlitebrowser	00:00:57		c55f4	TRUE	00:00:58
51	bed327	TRUE	00:01:53		sqlitebrowser	00:00:43		522d84	TRUE	00:01:37
52	d957e4	FALSE	00:01:19		html-pipeline	00:00:59		772988	TRUE	00:01:07
53	bed325	TRUE	00:02:23		sqlitebrowser	00:00:36		c55f44e	TRUE	00:00:37
54			00:00:51		sqlitebrowser	00:01:09		522d84	TRUE	00:00:49
55	77b4d5	FALSE	00:01:12		sqlitebrowser	00:00:40		c55f44e	TRUE	00:01:23
56	c1f9dae	FALSE	00:00:58		html-pipeline	00:00:39		d22605	FALSE	00:00:43
57	bed325	TRUE	00:01:22		sqlitebrowser	00:01:41		c55f44e	TRUE	00:01:08
58	944e02	FALSE	00:00:52		sqlitebrowser	00:01:37		c55f44e	TRUE	00:00:39
59	bed32	TRUE	00:01:24		sqlitebrowser	00:01:29		c55f4	TRUE	00:04:19
60	bed325	TRUE	00:01:23		sqlitebrowser	00:02:32		c55f44e	TRUE	00:01:07
61	944e02	FALSE	00:01:21		sqlitebrowser	00:02:08		772988	TRUE	00:00:49
62	bed325	TRUE	00:01:11		sqlitebrowser	00:01:44		c55f44e	TRUE	00:00:49
63	bed325	TRUE	00:01:12		sqlitebrowser	00:00:34		c55f44e	TRUE	00:00:51
64	bed325	TRUE	00:01:32		sqlitebrowser	00:01:27		c55f44e	TRUE	00:01:17
65	bed325	TRUE	00:01:47		sqlitebrowser	00:02:44		c55f44e	TRUE	00:00:53
66	77b4d5	FALSE	00:01:03		sqlitebrowser	00:00:39		b15607	FALSE	00:00:44
67	428940	FALSE	00:01:38		sqlitebrowser	00:00:50		c55f44e	TRUE	00:01:16
68	944e02	FALSE	00:01:11		sqlitebrowser	00:01:17		c55f44e	TRUE	00:00:43
69	a4be10	FALSE	00:00:36		sqlitebrowser	00:00:29		ec1f3a2	FALSE	00:00:39
70	73ef810	FALSE	00:02:19		vim.js	00:01:41		c55f44e	TRUE	00:00:51
71	bed325	TRUE	00:02:57		sqlitebrowser	00:01:07		522d84	TRUE	00:01:10
72	944e02	FALSE	00:01:02		sqlitebrowser	00:01:16		c55f44e	TRUE	00:00:44
73	bed325	TRUE	00:01:55		sqlitebrowser	00:01:46		c55f44e	TRUE	00:01:57
74	944e02	FALSE	00:00:43		sqlitebrowser	00:01:12		9c283d	FALSE	00:00:21

**Table B.5:** Raw results from the project comparison section (questions 11–13) in the user study with undergraduate students.

Red cells denote participant answers that do not match our ground truth for that question. Orange cells denote participant answers that took less than 10 seconds, and were ignored.

P	11	gt-agree	gt-disagree	time	12	12	html-pip	gt-agree	gt-disagree	time	13	gt-agree	gt-disagree	time
1	html-pipeline	BRA		00:02:31	html-pipeline	postr	TRUE	LAN		00:01:48	vim.js	AUT		00:01:32
2	html-pipeline	BRA		00:02:18	html-pipeline	postr	TRUE	LAN		00:02:53	postr		OTH	00:03:53
3	html-pipeline	BRA		00:01:38	postr	html-pipeline	TRUE	LAN		00:02:13	vim.js	LEN		00:02:34
4	html-pipeline	VIS		00:00:43	html-pipeline	postr	TRUE	VIS		00:00:45	vim.js	AUT		00:00:55
5	html-pipeline	BRA		00:01:29	html-pipeline	postr	TRUE	LAN		00:01:42	vim.js	AUT		00:02:11
6	html-pipeline	VIS		00:01:46	postr	html-pipeline	TRUE	VIS		00:03:30	vim.js	AUT		00:02:54
7	html-pipeline	BRA,VIS		00:03:19	html-pipeline	postr	TRUE	LAN,VIS		00:01:41	html-pipeline		LEN	00:07:27
8	html-pipeline	BRA		00:03:00	html-pipeline	postr	TRUE	LAN		00:00:54	vim.js	AUT		00:01:18
9	html-pipeline	BRA		00:01:54	html-pipeline	postr	TRUE	LAN		00:00:51	vim.js	AUT		00:00:38
10	sqlitebrowser	BRA		00:03:11	postr	html-pipeline	TRUE	LAN		00:02:03	vim.js	AUT		00:01:46
11	html-pipeline			00:01:02	html-pipeline	postr	TRUE	OTH		00:00:48	LightTable			00:00:12
12	html-pipeline	BRA		00:02:00	vim.js	LightTable	FALSE	LAN		00:01:57	vim.js	AUT		00:01:04
13	sqlitebrowser	BRA		00:02:07	vim.js	LightTable	FALSE	OTH		00:01:06	vim.js	LEN		00:01:25
14	html-pipeline	VIS		00:00:33	vim.js	LightTable	FALSE	VIS		00:00:24	sqlitebrowser			00:00:53
15	html-pipeline	BRA		00:03:28	html-pipeline	postr	TRUE	VIS		00:03:14	vim.js	LEN		00:04:57
16	html-pipeline	LIN		00:01:33	html-pipeline	postr	TRUE	LAN		00:01:41	sqlitebrowser		OTH	00:02:59
17	html-pipeline	BRA		00:02:46	html-pipeline	postr	TRUE	LAN		00:02:25	vim.js	AUT		00:01:39
18	html-pipeline	BRA		00:02:13	html-pipeline	postr	TRUE	LAN		00:01:06	vim.js	VIS		00:00:46
19	html-pipeline	LIN		00:01:07	html-pipeline	postr	TRUE	LAN		00:01:01	sqlitebrowser		OTH	00:01:14
20	sqlitebrowser	VIS		00:00:58	html-pipeline	postr	TRUE	VIS		00:00:51	vim.js	VIS		00:00:58
21	html-pipeline	BRA		00:01:42	postr	html-pipeline	TRUE	OTH		00:01:18	sqlitebrowser			00:01:22
22	html-pipeline	BRA		00:01:30	html-pipeline	postr	TRUE	LAN		00:01:08	LightTable		LEN	00:01:53
23	html-pipeline	BRA		00:01:43	html-pipeline	postr	TRUE	LAN		00:01:33	vim.js	AUT		00:02:04
24	html-pipeline	BRA		00:01:53	html-pipeline	postr	TRUE	LAN		00:01:48	vim.js	AUT		00:03:47
25	html-pipeline	VIS		00:02:08	html-pipeline	postr	TRUE	VIS		00:01:03	vim.js	AUT		00:01:28
26	html-pipeline	BRA		00:01:38	html-pipeline	postr	TRUE	LAN		00:01:44	vim.js	AUT		00:02:20
27	html-pipeline	BRA		00:00:57	html-pipeline	postr	TRUE	LAN		00:01:02	vim.js	AUT		00:02:21
28	html-pipeline	VIS		00:01:09	html-pipeline	postr	TRUE	VIS		00:01:15	sqlitebrowser		VIS	00:01:35
29	html-pipeline	BRA		00:01:05	html-pipeline	postr	TRUE	LAN		00:01:12	vim.js	LEN		00:01:03
30	html-pipeline	BRA		00:01:44	html-pipeline	postr	TRUE	LAN		00:01:06	vim.js	AUT		00:01:23
31	sqlitebrowser	VIS		00:00:49	sqlitebrowser	LightTable	FALSE	VIS		00:00:56	vim.js	AUT		00:00:45
32	html-pipeline	VIS		00:01:22	html-pipeline	postr	TRUE	OTH		00:01:45	vim.js	AUT		00:01:23
33	html-pipeline	VIS		00:00:53	html-pipeline	postr	TRUE	VIS		00:01:12	vim.js	LEN		00:01:40
34	html-pipeline	VIS		00:01:23	html-pipeline	postr	TRUE	LAN		00:01:28	vim.js	VIS		00:03:56
35	html-pipeline	VIS		00:01:10	LightTable	sqlitebrowser	FALSE	VIS		00:02:01	vim.js	AUT		00:02:30
36	html-pipeline	BRA,VIS		00:02:20	postr	html-pipeline	TRUE	LAN,VIS		00:02:08	vim.js	AUT		00:01:23
37	html-pipeline	VIS		00:01:40	html-pipeline	postr	TRUE	VIS		00:01:51	vim.js	AUT		00:01:28
38	html-pipeline	LIN		00:01:28	postr	html-pipeline	TRUE	OTH		00:00:56	vim.js	AUT		00:01:07
39	html-pipeline	VIS		00:01:07	LightTable	vim.js	FALSE	VIS		00:02:54	sqlitebrowser		OTH	00:03:59
40	sqlitebrowser	VIS		00:01:13	sqlitebrowser	html-pipeline	FALSE	VIS		00:01:14	vim.js	AUT		00:01:32
41	html-pipeline	LIN		00:01:54	html-pipeline	postr	TRUE	LAN,VIS		00:00:56	vim.js	AUT		00:01:22
42	html-pipeline	BRA		00:06:33	sqlitebrowser	LightTable	FALSE	LAN		00:02:16	vim.js	AUT		00:01:19
43	html-pipeline	VIS		00:01:06	html-pipeline	postr	TRUE	LAN		00:03:13	vim.js	AUT		00:02:41
44	html-pipeline	BRA		00:01:42	html-pipeline	postr	TRUE	VIS		00:02:23	vim.js	AUT		00:02:41
45	skipped				html-pipeline	postr	TRUE	LAN		00:00:19	sqlitebrowser		LEN	00:02:47
46	vim.js			00:03:43	sqlitebrowser	html-pipeline	FALSE	OTH		00:01:32	LightTable			00:00:53
47	html-pipeline	BRA		00:01:22	LightTable	vim.js	FALSE	LAN		00:01:39	vim.js	AUT		00:01:42
48	html-pipeline	BRA		00:01:54	html-pipeline	postr	TRUE	LAN		00:01:25	vim.js	AUT		00:02:22
49	html-pipeline	VIS		00:01:08	html-pipeline	postr	TRUE	VIS		00:01:11	vim.js	AUT		00:01:22
50	html-pipeline	BRA,VIS		00:03:44	html-pipeline	postr	TRUE	LAN		00:02:01	vim.js	LEN		00:01:30
51	html-pipeline	VIS		00:00:44	html-pipeline	postr	TRUE	VIS		00:00:39	skipped		AUT	
52	html-pipeline	VIS		00:01:13	html-pipeline	postr	TRUE	VIS		00:00:57	vim.js	AUT		00:01:12
53	html-pipeline	VIS		00:01:45	html-pipeline	postr	TRUE	LAN		00:01:12	vim.js	AUT		00:01:05
54	html-pipeline	VIS		00:00:50	html-pipeline	postr	TRUE	VIS		00:00:49	vim.js	AUT		00:01:04
55	html-pipeline			00:01:03	html-pipeline	postr	TRUE	OTH		00:00:30	vim.js	LEN		00:01:09
56	vim.js	BRA		00:01:15	html-pipeline	postr	TRUE	LAN,VIS		00:01:08	vim.js	AUT		00:01:00
57	html-pipeline	BRA		00:02:02	html-pipeline	postr	TRUE	LAN		00:02:21	vim.js	AUT		00:01:26
58	html-pipeline	BRA,VIS		00:01:06	html-pipeline	postr	TRUE	VIS		00:01:25	vim.js	AUT		00:01:43
59	html-pipeline	VIS		00:02:36	postr	html-pipeline	TRUE	VIS		00:00:54	vim.js	VIS		00:02:27
60	html-pipeline	BRA		00:01:37	html-pipeline	postr	TRUE	LAN		00:03:13	vim.js	AUT		00:02:11
61	html-pipeline	BRA,VIS		00:02:25	html-pipeline	postr	TRUE	LAN		00:01:35	vim.js	LEN		00:02:23
62	html-pipeline	BRA,VIS		00:01:08	html-pipeline	postr	TRUE	LAN		00:01:49	vim.js	AUT		00:01:49
63	html-pipeline	BRA		00:03:24	html-pipeline	postr	TRUE	LAN		00:02:01	sqlitebrowser		OTH	00:03:18
64	html-pipeline			00:02:41	html-pipeline	postr	TRUE	LAN		00:02:27	skipped			
65	html-pipeline	VIS		00:00:47	html-pipeline	postr	TRUE	LAN,VIS		00:01:09	vim.js	AUT		00:01:18
66	html-pipeline	VIS		00:01:22	vim.js	LightTable	FALSE	VIS		00:00:57	vim.js	LEN		00:01:00
67	html-pipeline	BRA		00:02:00	LightTable	vim.js	FALSE	LAN		00:02:00	sqlitebrowser		OTH	00:02:22
68	vim.js	BRA,VIS		00:02:04	html-pipeline	postr	TRUE	LAN		00:01:35	vim.js	AUT		00:01:46
69	html-pipeline	BRA		00:00:51	html-pipeline	postr	TRUE	LAN		00:01:31	vim.js	AUT		00:02:27
70	sqlitebrowser	VIS		00:00:52	html-pipeline	postr	TRUE	VIS		00:03:26	vim.js	VIS		00:01:49

P	11	gt-agree	gt-disagree	time	12	12	html-pip	gt-agree	gt-disagree	time	13	gt-agree	gt-disagree	time
71	html-pipeline	BRA		00:01:14	vim.js	LightTable	FALSE		LAN	00:01:07	vim.js	LEN		00:00:53
72	html-pipeline	BRA		00:01:24	html-pipeline	postr	TRUE	LAN		00:01:39	vim.js	AUT		00:01:21
73	html-pipeline	BRA		00:01:29	postr	html-pipeline	TRUE	LAN		00:01:30	vim.js	AUT		00:01:46
74	vim.js		VIS	00:01:43	sqlitebrowser	LightTable	FALSE		VIS	00:02:46	vim.js	VIS		00:01:41

**Table B.6:** Raw results from the exploratory question in the user study with undergraduate students.

Red cells denote participant answers that do not match our ground truth for that question.  
 Orange cells denote participant answers that took less than 10 seconds, and were ignored.

P	14-explain (raw text)	Metric 1	Metric 2*	Attrib	14	gt	gt-agree	gt-disagree	time
1	Commit message length	Commit Message Leng	~	Block Length	580455	FALSE		LIN,MSG	00:01:56
2	Commit Message Length	Commit Message Leng	?	?	7eddc5	FALSE		MSG	00:03:01
3	Commit Localization	Commit Localization	?	?	238dba	TRUE	MET		00:03:59
4	Most edited file	Most Edited File	~	Commit Message					~
5	Most edited files	Most Edited File	?	?	09039d	TRUE	MET		00:04:14
6	Message length	Commit Message Leng	?	?	be4b34	FALSE		LIN,MSG	00:03:14
7	commit message length	Commit Message Leng	~	Block Length	e07194	TRUE	LIN,MSG		00:03:57
8	Commit Message Length	Commit Message Leng	?	?	a430a	FALSE		MSG	00:01:43
9	Most edited file	Most Edited File	~	Block Length	09039d	TRUE	LIN		00:02:15
10	Commit message length	Commit Message Leng	~	Block Length	1557ac	TRUE	LIN,MET		00:03:20
11						FALSE			00:00:13
12	Commit message length	Commit Message Leng	~	Block Length	7390e4	TRUE	LIN,MSG		00:04:57
13									
14									
15	Commit message length, Mos	Commit Message Leng	~	Commit Message	1557a	TRUE	MET,MSG		00:04:02
16	Commit Message Length	Commit Message Leng	?	?	1557ac	TRUE	MSG		00:03:44
17	commit message length	Commit Message Leng	~	Block Length	1557ac	TRUE	LIN,MSG		00:05:05
18	most edited file	Most Edited File	?	?	eef9a3f	FALSE		MET	00:02:40
19	Message Length, Lines chang	Commit Message Leng	~	Block Length	c343fb	TRUE	LIN,MSG		00:03:29
20	Commit message length	Commit Message Leng	?	?	522d84	FALSE		MSG	00:02:29
21						FALSE			00:00:10
22	commit message length	Commit Message Leng	?	?	580455	FALSE		MSG	00:01:30
23	Commit Message Length	Commit Message Leng	~	Block Length	8f5b0	FALSE		LIN,MSG	00:02:44
24	Languages in a commit	Languages in a Commit	~	Commit Message	860100	FALSE		MET,MSG	00:04:15
25	Commit message length	Commit Message Leng	~	Block Length	1557ac	TRUE	LIN,MSG		00:02:13
26	Commit Localization	Commit Localization	?	?	5e1492	FALSE		MET	00:03:56
27	Commit Message Length	Commit Message Leng	~	Block Length	c343fb	TRUE	LIN,MSG		00:02:33
28	Commit message length	Commit Message Leng	~	Block Length	e07194	TRUE	LIN,MSG		00:04:19
29	commit message length	Commit Message Leng	~	Block Length	7390e4	TRUE	LIN,MSG		00:02:25
30	Commit Message Length	Commit Message Leng	?	~	7390e4	TRUE	LIN,MSG		00:02:31
31	Commit Message Length	Commit Message Leng	?	?	e07194	TRUE	MSG		00:01:38
32	Commit Message Length	Commit Message Leng	~	Block Length	9c283d	FALSE		LIN	00:02:32
33	most edited file	Most Edited File	~	Commit Message	09039d	TRUE	LIN,MSG		00:03:40
34	commit localization	Commit Message Leng	Commit Localization	~	8f5b0e	FALSE		LIN,MET	00:02:29
35	Commit Message Length	Commit Message Leng	~	Block Length	9c283d	FALSE		LIN,MSG	00:02:26
36	Localization	Commit Localization	~	Commit Message	238dba	TRUE	MET,MSG		00:04:47
37	Languages in a commit	Languages in a Commit	~	Commit Message	87eff91	FALSE		MET,MSG	00:03:11
38	Most edited file	Most Edited File	?	?	09039d	TRUE	MET		00:02:21
39	Commit Localization	Commit Localization	~	Commit Message	9c283d	FALSE		MSG	00:04:06
40	Commit Localization	Commit Localization	?	?	1557ac	TRUE	MET		00:02:54
41	Commit Localization	Commit Localization	~	Commit Message	58ab0d	FALSE		MET,MSG	00:05:49
42	The biggest whitest block	Commit Message Leng	~	Block Length	7390e4	TRUE	LIN,MSG		00:02:52
43	Commit Message Length	Commit Message Leng	?	?	e07194	TRUE			00:03:02
44	commit message length	Commit Message Leng	?	?	7390e	TRUE	MSG		00:02:35
45	Commit message length	Commit Message Leng	?	?	c343fb	TRUE	MSG		00:04:24
46	block length	?	?	Block Length	7390e4	TRUE			00:04:34
47	Commit message length and l	Commit Message Leng	~	Block Length	09039d	TRUE	LIN,MSG		00:02:23
48	Commit Localization	Commit Localization	?	?	3748e4	FALSE		MET	00:03:22
49	most edited file	Most Edited File	~	Commit Message	b8d043	FALSE		MET,MSG	00:03:13
50	Commit Message Length	Commit Message Leng	~	Block Length	7390e	TRUE	LIN,MSG		00:02:36
51	color	?	?	?	4ab142	FALSE		MSG	00:03:11
52	Commit Message Length	Commit Message Leng	?	?	c343fb	TRUE	MSG		00:02:18

P	14-explain (raw text)	Metric 1	Metric 2*	Attrib	14	gt	gt-agree	gt-disagree	time
53	Commit Message Length	Commit Message Leng	?	?	e07194	TRUE	MSG		00:02:15
54	fix buils	?	~	Block Length	fe9105	FALSE		MET,MSG	00:03:57
55	most edited file	Most Edited File	?	?	09039d	TRUE			00:01:50
56	commit message length	Commit Message Leng	?	?	c343fb	TRUE	MSG		00:02:21
57	Commit message length	Commit Message Leng	~	Block Length	c343fb	TRUE	LIN,MSG		00:03:29
58	Length of block and color	Commit Message Leng	~	Block Length	7390e4	TRUE	LIN,MSG		00:02:40
59	Commit Message Length	Commit Message Leng	~	Block Length					
60	Commit Localization, Commit	Commit Message Leng	Commit Localization	~	238dba	TRUE	MET,MSG		00:09:56
61	commit message length	Commit Message Leng	?	?	c343fb	TRUE	MSG		00:04:19
62	Most Edited File	Most Edited File	?	?	eef9a3f	FALSE		MET	00:02:21
63	Most files edited	Most Edited File				FALSE			00:02:49
64									00:02:45
65	Commit Metric Length	Commit Message Leng	~	Block Length	c343fb	TRUE	LIN,MSG		00:03:34
66	Block Length	?	?	Block Length	1557ac	TRUE	LIN		00:01:08
67	Message Length	Commit Message Leng	~	Block Length	238dba	TRUE	LIN,MSG		00:02:41
68	commit message length	Commit Message Leng	?	?	580455	FALSE		MSG	00:02:01
69	commit localization	Commit Localization	~	Commit Message	3748e4	FALSE		MET,MSG	00:02:20
70	color	?	?	?	ce8a4f	FALSE		MET	00:03:33
71	Most Edited File	Most Edited File	?	?	85985d	FALSE			00:02:25
72	Commit Localization	Commit Localization	~	Commit Message	c55f44	FALSE		MET,MSG	00:10:22
73	message length	?	~	Message Length	7390e4	TRUE	MSG		00:03:32
74	commit message length	Commit Message Leng	?	?	1557ac	TRUE	MSG		00:01:54

- **VIS** The explanation discusses the visual characteristics of the repository footprints. e.g., “Similar visualization”, “Similar pattern”, “Similar colors”
- **BRA** The explanation discusses the branches, based on the values presented by the *branches used* or *number of branches* metrics. e.g., “Increased number of branches over time”, “Similar number of branches”
- **LIN** The explanation discusses the number of LoC changed, based on the length of commit blocks when the appropriate block length mode was selected. e.g., “High number of lines changed towards the end”, “Larger commits towards the end”
- **LAN** The explanation discusses the number of distinct programming languages involved in a commit, based on the *languages in a commit* metric. e.g., “Both projects use few languages”
- **AUT** The explanation discusses the scripted/automated commits performed on the **vim.js** project
- **LEN** The explanation discusses the difference in commit message length, based on either the *commit message length* metric or based on reading the actual commit message on the commit block
- **MSG** The explanation discusses the contents of the commit message of the commit that was selected by the participant in the exploratory question
- **MET** The explanation discusses the metric values of the metric that was selected by the participant in the exploratory question
- **OTH** Other types of explanations or unclear explanations



## Appendix C

# Software engineering researchers study

This appendix contains meta-data and raw results for the user study with SE researchers described in Section 5.2.

### C.1 Protocol and questionnaire

#### C.1.1 Procedure overview

The focus of the study is to show that participants can use RepoGrams for pattern identification. We want to focus the study on finding clusters/categories in the repositories.

The study will be conducted individually with each participant, either in a meeting room or over the web with a video conference software.

#### C.1.2 Study protocol

We begin the description of each question with the following data:

[metric1, metric2, ..., metricN ; metrics-grouping ; block-length ; repo-url1, repo-url2, ..., repo-urlN]

This lists the various settings that RepoGrams will be set to prior to each question: which metric(s) will be used, which grouping mode, which block length

mode, and what repository URLs.

For the preparation and main tasks we will notify the participants a minute before they are running out of time. When the participant runs out of time we will ask them to mark “skipped” on the question and skip to the next question. At any point we will give the participants the option of skipping a task that they cannot complete. We will use the pilot studies to estimate how long the tasks should take.

The participants will answer the tasks using a custom online questionnaire software.

### **C.1.3 Questionnaire**

#### **Demographics**

a. Gender

- Male
- Female
- Other/rather not say

b. How often do you use Distributed Version Control Systems?

- Never/Rarely
- Once a month
- Once a week
- Daily

c. What is your academic status?

- Undergraduate student
- Masters student
- PhD student
- Postdoc
- Faculty

d. *[multiple choice]* As part of your research, have you performed any of the following?

- Studied a version controlled project repository
- Inspected the evolution of a software project
- Evaluated a tool using artifacts (e.g., source code, logs, bug tracking issues) from one or more software projects

### **Introduction to RepoGrams**

We will now introduce RepoGrams and demonstrate the tool for you.

At the beginning of the study we will introduce and demonstrate the tool to the participant, taking approximate 15 minutes. The rest of this section was not part of the questionnaire itself, but rather a summary of the topics that were covered during the demonstration as part of this section of the questionnaire.

#### 1. Big-picture description:

- What does the tool do: RepoGrams visualizes information about git repositories. It takes git repositories, lays all the commits out onto a single horizontal line as **blocks**, and colors each block a color that correspond to a value of some chosen **metric**. A metric can represent information from a variety of domains, such as features of the code (e.g., number of classes added/removed/modified) or of the development process (e.g., who made the commit). The tool helps the tool user investigate some aspects of git repositories or compare various features of the selected projects.
- Imagine that you developed some SE tool and you want to write a paper on that tool. Before you can do that, you have to run an evaluation on it, and before you can run an evaluation you need to choose projects to evaluate your tool on. We conducted a literature survey of SE papers and found that researchers rarely explain how they chose their evaluation targets or why they chose this subset and not another that would fit their criteria. We created the tool as a response to this problem, we

hope to show that RepoGrams can be used to help choose projects with stronger confidence.

- Our main target audience is SE researchers, but the tool can also potentially be used by others, such as project leaders, managers, etc. . .

2. Basic metaphor: [in this part, as we go through the concepts we will demonstrate them using the tool]

- The basic metaphor of RepoGrams is as follows: Each line represents a single git project as it is represented in a single metric.
- Each block represents a single commit, and the commits are laid out in temporal order, regardless of parent commit, from left to right. So the first commit in the project is at the far left and the latest commit is on the far right. Between the different metrics the same block represents the same commit, only in that different metric.
- The block length can be changed to represent different things: it can represent how many lines of code have been changed in the commit (called churn in git), either comparable between the projects or incomparable (to see an overview of all projects), or it can just be a fixed width if the churn does not matter.
- The color of each commit represents its value in this specific metrics (see legend, description of metric)

3. Basic interactions available in the UI: [in the section we guide the participant with step-by-step instructions to play around with the tool]

- Demonstrate adding 3 repositories
- Demonstrate zoom and scroll
- Demonstrate changing metrics, changing block length
- Demonstrate swapping repository order, removing repository
- Demonstrate loading of example data

4. Let the participant ask questions about the interface and suggest that they take a minute to try it themselves. Explain that next up we will give them 3 tasks to perform, and that during those three tasks they can ask us questions, but after those 3 tasks we will not be able to answer any question about the interface or the tasks (except for clarifications if the instructions are unclear)

### Preparation tasks

This section includes 3 tasks. You can spend up to 5 minutes to finish these tasks (we will start the timer when you switch to the next page).

While working on these tasks you may ask clarification questions. Note that in the *next* section we will not be able to answer any questions regarding the interface.

#### 1. Preparation task 1

- [POM Files ; metrics-first ; fixed-length ; <https://github.com/sqlitebrowser/sqlitebrowser>, <https://github.com/coolwanglu/vim.js>, <https://github.com/mattgallagher/AudioStreamer>, <https://github.com/LightTable/LightTable>, <https://github.com/jch/html-pipeline>]
- Based on the current view, what is your estimate of the number of commits in the AudioStreamer project?
  - 1–9
  - 10–99
  - 100–999
  - 1,000–9,999
  - 10,000–99,999
- (Based on question 2 from the undergraduate study)

#### 2. Preparation task 2

- [Commit Localization ; metrics-first ; fixed-length ; <https://github.com/sqlitebrowser/sqlitebrowser>, <https://github.com/coolwanglu/vim.js>, <https://github.com/LightTable/LightTable>, <https://github.com/jch/html-pipeline>, <https://github.com/GNOME/postr>]

- Based on the current view, which project had the longest uninterrupted sequence of highly localized (0.881.00) commits?
  - sqlitebrowser
  - vim.js
  - LightTable
  - html-pipeline
  - postr
- (Based on question 9 from the undergraduate study)

### 3. Preparation task 3

- [Number of Branches ; metrics-first ; lines-changed-incomparable ; <https://github.com/sqlitebrowser/sqlitebrowser>, <https://github.com/coolwanglu/vim.js>, <https://github.com/LightTable/LightTable>, <https://github.com/jch/html-pipeline>, <https://github.com/GNOME/postr>]
- Based on the current view, which project appears to have a development process that is most similar to LightTable?
  - sqlitebrowser
  - vim.js
  - LightTable
  - html-pipeline
  - postr
- (Based on question 11 from the undergraduate study)

### Main tasks

#### *Last change for questions*

Please take a bit of time to explore RepoGrams and ask clarifying questions. Beyond this point we will not be able to answer questions regarding the interface, the definitions of the metrics, and other elements of the tool.

#### 4. Main task 1

This task has a 3 minutes time limit.

- [Most Edited File ; metrics-first ; fixed-length ; <https://github.com/phusion/passenger-docker>]
- Based on the current view, which of the following is true?
  - There is a general UPWARDS trend to the metric values
  - There is a general CONSTANT trend to the metric values
  - There is a general DOWNWARDS trend to the metric values

#### 5. Main task 2

This task has a 5 minutes time limit.

- [POM Files ; metrics-first ; fixed-length ; <https://github.com/facebook/css-layout>, <https://github.com/qiujuer/Genius-Android>, <https://github.com/JakeWharton/butterknife>, <https://github.com/AndroidGears/Plugin>, <https://github.com/pedrovg/TuentiTV>, <https://github.com/ksoichiro/Android-ObservableScrollView>, <https://github.com/square/picasso>, <https://github.com/google/iosched>, <https://github.com/square/retrofit>]
- Categorize the projects into two clusters — one cluster containing projects that use Maven (include .pom files), and the other cluster with projects that do not use Maven
- Note: Our online questionnaire included a subquestion here. However, during our postmortem we found that the question was understood ambiguously by a large number of participants. As a consequence we removed that subquestion from our analysis and do not report on it here.

#### 6. Main task 3

This task has a 5 minutes time limit.

- [Branched Used ; metrics-first ; fixed-length ;  
<https://github.com/munificent/wren>,  
<https://github.com/PHPMailer/PHPMailer>,  
<https://github.com/yahoo/pure>,  
<https://github.com/stympy/faker>,  
<https://github.com/mmozuras/pronto>]
- Categorize the projects into two clusters — one cluster containing projects that were developed on a single master branch before branching off to multiple branches, and the other cluster containing projects that branched off early in their development

#### 7. Main task 4

This task has a 5 minutes time limit.

- [Commit Author, Branches Used ; repos-first ; lines-changed-incomparable ;  
<https://github.com/JedWatson/touchstonejs>,  
<https://github.com/pblittle/docker-logstash>,  
<https://github.com/lukasschwab/stackit>,  
<https://github.com/arialdomartini/oh-my-git>]
- Categorize the projects into two clusters — one cluster containing projects that have a correlation between branches and authors, and the other cluster with projects that do not exhibit this correlation

#### 8. Main task 5

This task has a 7 minutes time limit.

- [Commit Author ; repos-first ; lines-changed-comparable ;  
<https://github.com/lukasschwab/stackit>,  
<https://github.com/deployphp/deployer>,  
<https://github.com/sequenceiq/docker-ambari>]
- (a) Categorize the projects into two clusters — one cluster has projects that have one single obvious dominant contributor, based on number of **LINES OF CODE CHANGED**, and the second group does not have such a contributor. A dominant contributor is one that generated close to or more than 50% of the line changes in the project.  
 Hint: you might have to zoom in and scroll.
- **Before the next question, switch to the Fixed block length mode**



- (b) Categorize the projects into two clusters — one cluster has projects that have one single obvious dominant contributor, based on number of **COMMITTS**, and the second group does not have such a contributor. A dominant contributor is one that generated close to or more than 50% of the commits in the project.

Hint: you might have to scroll.

### **Open-ended questions**

- Do you see RepoGrams being integrated into your research/evaluation process? If so, can you give an example of a research project that you could use/could have used RepoGrams in?
- What are one or two metrics that you wish RepoGrams included that you would find useful in your research? How much time would you be willing to invest in order to write code to integrate a new metric?
- In your opinion, what are the best and worst parts of RepoGrams?
- Choose one of the main tasks that we asked you to perform. How would you have performed it without RepoGrams?
- Do you have any other questions or comments?

### **C.1.4 Filtering results**

We discarded the results from 1 participant who was disqualified for not having any prior experience with repository analysis (i.e., said participant did not check any boxes in Demographics question d.)

## **C.2 Raw results**

Here we list the raw results. The difference background colors for cells denote equivalence sets. Each equivalence set denotes how many participants responded with the same answer for each question.

**Table C.1:** Raw results from the user study with SE researchers.

Demographics				Preparation questions			Main questions							
P	a	b	c	d	1	2	3	4	5	6	7	8a	8b	
1	M	Daily	Faculty	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	YY	
2	M	Once a week	Masters student	Y,Y,Y	100-999	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
3	M	Once a week	PhD student	Y,Y,Y	10-99	sqlitebrowser	vim.js	constant	NNNNNNYY	LEELL	YNY	NY	NY	
4	M	Once a week	Faculty	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
5	F	Once a month	Postdoc	Y,Y,N	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
6	M	Daily	PhD student	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	YY	
7	M	Once a week	Faculty	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
8	F	Once a week	PhD student	N,N,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
9	M	Once a week	PhD student	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
10	M	Daily	Masters student	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	downwards	NNNNNNYY	LEELL	YNY	NY	NY	
11	M	Once a week	Faculty	N,N,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	YY	
12	M	Once a week	PhD student	Y,N,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
13	M	Daily	Faculty	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
14	M	Daily	PhD student	Y,Y,Y	10-99	sqlitebrowser	html-pipeline	upwards	NNNNNNYY	LEELL	YNY	NY	NY	
Percentage of participants who chose the most common answer					92.86%	100.00%	92.86%	85.71%	92.86%	85.71%	42.86%	78.57%	64.29%	

We do not report raw text responses to preserve the anonymity of the participants.

# Appendix D

## Case study

This appendix contains meta-data and raw results for the case study to estimate the effort involved in adding new metrics, described in Section 5.3.

### D.1 Results

#### D.1.1 Overview

We conducted a case study to estimate the effort involved in integrating new simple metrics into RepoGrams with two developers.

Dev1 is a computer science masters student who is the author of this thesis.

Dev2 is a computer science fourth year undergraduate student.

Both developers were not familiar with the codebase before they started the experiment. Dev1 added 3 metrics between running the undergraduate user study and running the user study with SE researchers, to be used in the latter user study. Dev2 added 3 metrics that were requested by some of the participants from the user study with SE researchers. Dev1 shared only the following details with Dev2:

- Development environment setup instructions
- The names of the 3 metrics that Dev1 added, to be used as starting points for the exploration of the codebase

Both developers proceeded in a linear fashion, starting with setting up the development environment, then exploring the code, and finally developing, integrating, and testing each metric individually. Both developers timed themselves as they performed each step.

### D.1.2 Raw results

Development environment setup time:

- Dev1: 20 minutes
- Dev2: 39 minutes

Exploration of the codebase:

- Dev1: 10 minutes
- Dev2: 40 minutes

Metrics implementations<sup>1</sup>:

**Table D.1:** Raw results from the case study to estimate the effort involved in the implementation of new metrics.

Dev	Metric name	Time (min)	LoC Python	LoC JavaScript	LoC JSON	LoC HTML & CSS
Dev1	POM Files	30	7	7	16	0
Dev1	Commit Author	52	12	26	730	33
Dev1	Commit Age	48	8	30	50	0
Dev2	Files Modified	42	7	7	16	0
Dev2	Merge Indicator	44	8	7	16	0
Dev2	Author Experience	26	17	7	16	0

<sup>1</sup>The number of LoC changed reported in Table D.1 might differ from those reported in Section 5.3. The codebase for RepoGrams was refactored between the case study and the writing of this thesis as a result of the case study to facilitate the addition and implementation of new metrics. As part of the refactoring process, many metrics were rewritten to take advantage of these changes.

## **Appendix E**

### **License and availability**

RepoGrams is free software released under the GNU/GPL License [26]. The source code for RepoGrams is available for download on GitHub [53]. A running instance of RepoGrams is available at <http://repograms.net/>.