Orion: A Software Project Search Engine with Integrated Diverse Software Artifacts

Tegawendé F. Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, Laurent Réveillère

Orion: A Software Project Search Engine with Integrated Diverse Software Artifacts

Tegawendé F. Bissyandé[1], Ferdian Thung[2], David Lo[2], Lingxiao Jiang[2] and Laurent Réveillère[1]
[1]Laboratoire Bordelais de Recherche en Informatique, France
[2]Singapore Management University, Singapore
bissyande@labri.fr, ferdianthung@smu.edu.sg, davidlo@smu.edu.sg, lxjiang@smu.edu.sg, reveillere@labri.fr

***Abstract*—Software projects produce a wealth of data that is leveraged in different tasks and for different purposes: researchers collect project data for building experimental datasets; software programmers reuse code from projects; developers often explore the opportunities for getting involved in the development of a project to gain or offer expertise. Finding relevant projects that suit one needs is however currently challenging with the capabilities of existing search systems. We propose *Orion*, an integrated search engine architecture that combines information from different types of software repositories from multiple sources to facilitate the construction and execution of advanced search queries. Orion provides a declarative query language that gives to users access to a uniform interface where it transparently integrates different artifacts of project development and maintenance, such as source code information, version control systems metadata, bug tracking systems elements, and metadata on developer activities and interactions extracted from hosting platforms. We have built an extensible system with an initial capability of over 100,000 projects collected from the web, featuring several types of software repositories and software development artifacts. We conducted an experiment with 10 search scenarios to compare Orion with traditional search engines, and explore the need for our approach as well as the productivity of the proposed infrastructure. The results show with strong statistical significance that users find relevant projects faster and more accurately with Orion.**

I. Introduction

Software programmers, managers, and researchers often have the need to search for software projects for various reasons, such as looking for good pieces of code to reuse, looking for a good project to join, looking for good subjects for analysis, etc. However, their activities are often challenged by the difficulties in identifying appropriate software projects satisfying their needs. For example, in software engineering research, developing evidences to validate one's results often requires empirical evaluation with real-world datasets that are collected on the basis that they hold properties relevant to the evaluation. To this end, researchers usually resort to existing software project repositories where large amounts of data are publicly available for use to answer an enquiry like "*what projects produce source code with more than 1,000 test cases and where more than 10,000 bugs have been reported?*". Unfortunately, identifying and collecting such data, while lacking any scientific value, is an obligatory passage which is commonly recognized to be a tedious exercise [7]–[9]. Indeed, the dispersion of datasets in various repositories as well as the heterogeneity of their representations complicate extensive analysis of data from multiple sources. Furthermore, extracting knowledge from the large sets of information is challenging as it requires time, computational power, and an understanding of the correlations that exist among the data. Consequently, researchers who undertake to collect software data for their studies usually spend a significant portion of their time searching across different platforms to look for the few software systems that meet their requirements. Simplifying software data collection and manipulation as well as improving navigation across those data may save much research time and yield a positive impact on the strength and the validity of research studies.

The aforementioned challenges also apply to developers and users. For example, a developer wishing to participate in collaborative development projects can hardly determine beforehand which project would benefit the most from his expertise while being a "friendly" environment for him. It is indeed challenging to answer, e.g., to the question *What projects are developed in Ocaml by less than 10 people who have produced over 100,000 lines of code and are still maintaining the code with a high-bug fixing rate?* A user in a quest for the "right" software may also experience the frustration of settling for the *less relevant* ones that web search engines may lead him to.

In this work, we propose *Orion*, a search engine architecture where information extracted from source code, versioning repositories, bug tracking systems, collaborative development platforms, etc. are *merged* to build a knowledge database that can easily be queried through a user-friendly language, the Orion DSL. To this end, we have acquired and curated large amounts of data from a number of popular project hosting platforms. We have then defined and isolated a number of software system characteristics and their corresponding development artifacts that we have furthermore *linked* to enable advanced queries that may span over multiple information sources.

Orion combines the data sources and integrates the information so as to allow the linked data to be searched in a unified platform. Related work to ours has mainly focused on making the data readily available by means of centralization [1], [9], or aimed at regularly delivering snapshots and statistics on monitored projects [12]. While these efforts have enabled researchers to more easily study the data globally, they do not expose any new combinations of data to yield richer datasets, nor do they allow for advanced queries across data sources. We extend these work by building a search engine on top of a knowledge database. A more recent work by Dyer *et al.* proposes the BOA [6] infrastructure for mining software repositories. Their approach however is targeted for computing

statistics across software repositories, while Orion aims at facilitating the selection of a set of software projects satisfying various criteria based on properties of their development artifacts (e.g., source code size, programming languages, bug fix rate, team size, etc.). BOA only analyzes information available on SourceForge. Our approach however integrates information from various sources including GitHub, Google Code, JIRA, etc.

The main contributions of this paper are as follows:

- We discuss the opportunities and challenges in the context of project search and propose the Orion integrated search engine for empowering users, developers and researchers in their quests for relevant software projects.
- We explore the design of the Orion system, including the DSL, the mapping of raw data to domain-specific types, and the crawling step of project data for translation into a common format.
- We highlight, with a few query examples, the capabilities of our prototype implementation with datasets from a large variety of sources, such as bug reports, source code versioning information, and developer activities, from tens of thousands software projects. We also provide an assessment based on a user-study for estimating the need, the usability and effectiveness of Orion.

The rest of this paper is organized as follows. Section II motivates this work, by illustrating the need for an integrated search engine, and by outlining the challenges that we face in its design. Section III discusses our approach through a presentation of the design of the Orion DSL and an overview of the construction of Orion knowledge database. In Section IV, we propose an assessment of the Orion approach. Finally, Sections V and VI discuss related work and conclude with future work respectively.

II. Motivations

Software projects data are prevalent on the world wide web. Thanks to the momentum of open source philosophy, a wide range of project development artifacts are available on project homepages and on hosting platforms. The availability of such data has created unprecedented opportunities for research studies on real software development activities, for code reuse by novice as well as experienced programmers, for the exploration of diversified software alternatives, etc. Discovering the relevant project however, remains a tedious endeavor, with the current search capabilities on the world wide web. In this section we describe two complex and challenging search scenarios for different kinds of software project seekers. We subsequently discuss the requirements for improving the search for software projects.

A. Search Scenarios

A researcher on a quest for datasets. Research fellow Robert is interested in bug localization. To assess his latest approach, he is in need for datasets from several programs with specific development properties: (1) The programs must have had a significant number of bugs during their development cycle; (2) the programs should include test cases as, commonly, test cases allow to detect many bugs and thus can be used to replay software faults; (3) finally, to reduce the threats to validity and ensure that his approach can be generalized, Robert is seeking for programs from different software development teams.

Unfortunately, this search scenario requires information that is dispersed and may even be hidden in the deep web (e.g., in the source code). Thus, to retrieve such a set of programs, Robert must visit numerous websites, potentially downloading several software programs that will later be dismissed, before settling for a set of datasets that is smaller than he hoped for or whose development artifacts do not properly fit Robert's goals.

A programmer searching for a good and suitable project to reuse. Mark is a programmer whose latest assignment involves the implementation of a visualization software. He has opted for OpenGL, but for bootstrapping reasons, he is seeking code samples for understanding the internals of the API and potentially for reuse. Mark thus seeks (1) a project dealing with OpenGL, that is (2) still actively maintained and that (3) other people have tested before and contributed with feature requests and bug reports. In addition, Mark has an obligation to (4) respect his company restrictions on licenses.

As in the case of the previous scenario, using the traditional approach, this search is challenging. It consists of scanning programmer forums, and visiting several development websites. To address these challenges, we build an extensible and maintenable knowledge database of software projects that will deliver rich query capabilities for users.

B. Summary of Challenges

From the scenarios depicted above we distinguish two main challenges for an effective search of software projects on the web: (1) the myriads of information that make a software project unique are not maintained together, complicating the construction of queries that simultaneously considers multiple criteria; (2) the capabilities of traditional web search engines are limited in the exploitation of the semantics of user queries that are simply treated as a series of terms, and each term as a series of symbols from an alphabet.

1) Dispersed Information: Information on development artifacts of software projects are stored in heterogeneous repositories which are often located at various decorrelated which, furthermore, may not be connected among them: e.g., developers commit code changes in version control systems, report and manage bugs in bug trackers, discuss and interact in mailing lists or hosting platforms, etc. This dispersion of data complicates any search that leverages a combination of criteria. Even on hosting platforms, such as GitHub[1] and Freecode[2], where these information appear together in the *project page*, they are still not integrated in a way that allows a search using criteria based on multiple information types.

2) Flat Queries: Traditional search engines like the Google™ search engine, index web pages based on the appearance of terms and rely on information retrieval techniques to return query results. Consequently, a project seeker who enters

[1] github.com [2] freecode.com

a flat query, e.g., "project in Ocaml", can only expect search engines to return pages containing the provided keywords without any guarantee that the terms appearing in the pages bear the same meaning as intended by the user. Thus, while having been proven powerful and scalable, current search engines are simply not *data aware* [3]. Indeed, in such settings, the notion of a *project entity* does not exist and meaningful project query results cannot be returned where all attributes are composed to form the relevant data.

C. Search Requirements

To address the challenges for finding relevant software projects, two essential requirements must be met. We now review them to motivate the design of Orion.

Information Integration

The first step for improving the effectiveness of project search should consist in limiting the impact of the heterogeneity of software repositories as well as resolving the difficulties imposed by the dispersion of the data in various locations. Figure 1 illustrates the information types that should be leveraged to process the queries for the previously described search scenarios.

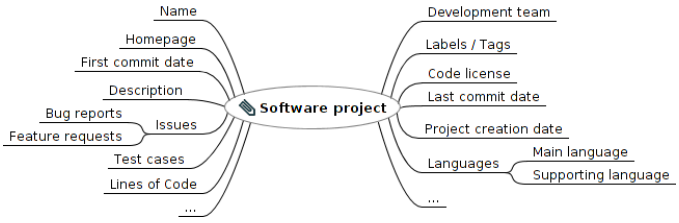

Fig. 1. Information types in a software project's data

We detail in the following the variety of information that are mentioned in the above figure to infer the terms and types that can be used to build the semantics of project search queries:

✓ *Project metadata*: A software project is initiated by an individual/company, often referred to as the **owner**, and may be supported by an **organization** (e.g., the Apache Software Foundation). These types of information are important as they can be used to infer the developing community and the programming and management styles that are prevalent in the project development. A **description** is also useful to classify the object of a project and the scope of its products. Information on the project development **homepage** and its **url** on a hosting platform may also be of interest to a project seeker. Finally, information on a software project may include **labels** which indicate the application domains, the technologies used, the main concepts developed, etc.

Project metadata are types of information which can be found on a project page.

✓ *Source code information*: Software projects produce programming code written using a programming **language** among the hundreds that have been designed over the years [19]. Often, the project contains a **main language** which represents the largest portion of the total **lines of code** (loc), but may also include several **supporting languages** such as the scripting languages that are used for automating compilation and testing tasks. Furthermore, the structure of the source code may reveal the presence of **test cases** (tests), i.e., program files that are directly included by developers to incrementally test the core code in order to check the accordance of the implementation with design specifications or to ensure backward compatibility as the project evolves.

Source code information being buried in the raw contents of project programs, retrieving them require to download and process large amounts of data.

✓ *Bugs and features management*: Software development cycles are rhythmed by the **issues** to address. Users often file **bug reports** (bugreps) to help improve code quality of file **feature requests** (featreqs) to contribute to project enhancement. Information on the volume of such data for a given project can provide insights on the maturity of a project as well as its development status, i.e., whether developers still improve the products or whether it is no longer maintained.

Project maintainers often setup issue trackers to record and manage issues submitted by software users.

✓ *Development context*: Finally, a software project is an evolving ecosystem. Besides the **creation date**, important information on the activities in a project include the **last commit date**, i.e., the date at which a code change was last introduced. The size of the development **team** is also an important criterion for differentiating projects, as the number of **authors** involved in the project may have an impact on the quality of the code base, both in terms of functionalities and of bug management.

Information on the development context can be retrieved from version control systems such as *git*[3] or *subversion*[4] which are used to setup project code repositories.

Information integration requires the collection of all the previously described information types to create a *project entity* whose instances can be queried with parameterized criteria. Indeed, the value and usefulness of each information type increases when it can explicitly be linked with related information types. This linking step must be preceded by other steps including the identification of software projects, the extraction of relevant metadata and processing of these data to build structured entities describing projects in their global representations. These steps however are challenged by the overwhelming amounts of data, the variety of their representations and the multiplicity of their locations on the web. Nonetheless, once these steps are completed, it becomes possible to compose meaningful queries that correlate different information types to retrieve an appropriate set of results.

Towards semantic query of software projects.

The notion of *semantic query* generally suggests that a search engine implements techniques that allow it to return results to a query based on what it believes the searcher is intending to find. Users may thus provide their queries in natural language and the engine may attempt to answer the query directly instead of returning urls suggesting web pages where the information *might be hidden*.

[3] git-scm.com [4] subversion.tigris.org

Towards implementing semantic query, a projects search engine should (1) deal with the processing and collection of domain specific information that are beyond string literals contained in web documents, and (2) propose languages for expressing domain specific user queries to find relevant software projects that fit a user/developer/researcher needs.

Concretely, to enable an effective search of software projects, a more expressive query system must be devised, possibly with some support for semantic querying. Indeed, once all information related to a software project are integrated, a project seeker should be empowered to construct rich queries that exploit the diversity of information types to accurately identify projects that are relevant to his needs. Where traditional search engines cannot uncover the "intended meaning" of *flat queries*, a dedicated project search engine should capture the *semantics* of structured and flexible queries.

To meet these requirements, we have designed and implemented the Orion integrated search engine. Orion includes an extensible and maintainable knowledge database of software projects with information retrieved from a variety of sources, and is complemented with a declarative language for allowing project seekers to compose expressive queries to easily search across the database.

III. ORION

Figure 2 illustrates the overall approach of Orion for addressing the challenges in searching software projects. The main objective of Orion is to reduce the gap between user's *search intent* and the interpretation that search engines associate to search queries. To this end, we propose to use a domain-specific language to formulate search specifications that describe the criteria for selecting relevant projects. The queries written with the high-level Orion DSL are translated into low-level advanced database queries where several information tables are joined to select, from a knowledge database, a set of projects. This knowledge database of projects is constructed and maintained by crawling the web for project code and development artifacts. In the following, we provide implementation details of these aspects of Orion.

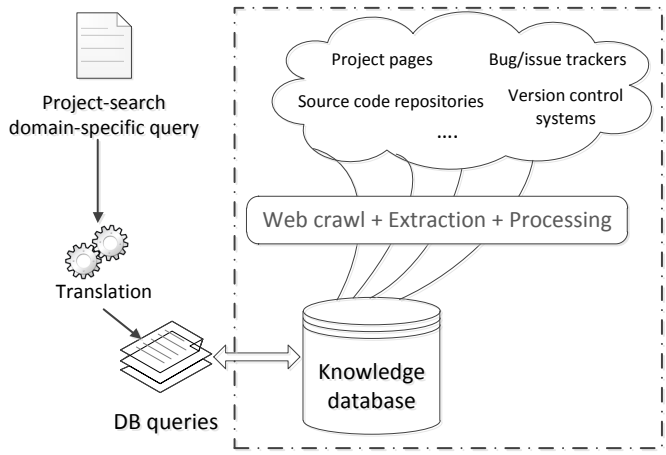

Fig. 2. The Orion approach

A. A Declarative Query Language

Ideally, project seekers should be enabled to enter complex queries in natural language which would be processed to extract the semantics of search requests in order to assess whether these can be served by the underlying database. However, the value-added of such elaborate engines is disproportionate compared to the implementation efforts that are required to deliver such services for the very narrow domain of project search. Nonetheless, allowing expressive and intuitive queries remain essential for improving the user-experience in the quest for software projects.

Towards supporting semantic query for searching software project instances, we propose a domain-specific language (DSL) which is built to capture the characteristics of software projects, the variety of search criteria, and the readiness at which users can navigate across project data.

The proposed language is designed with the intent of addressing problems that are relevant to the search of software projects. Consequently, this DSL aims at providing notations and abstractions with an expressive power focusing in this restricted domain. The design of the DSL was undertaken after a domain analysis based on the search scenarios discussed in Section II-A. We target different categories of users with various tolerance levels for complexity and learning curves. As every users would prefer a language that is intuitive and yet expressive enough to express various needs, we consider, for the design of the Orion DSL, a user-friendly syntax that is close to natural language and a precise semantics that is mapped on common project information types.

Syntax. The syntax of a language defines what keywords are used in the language and how users can compose those keywords to form *sentences*. In the case of project search, we have opted for a DSL close to *human natural* language using common English keywords and expressions (e.g., "SEARCH 3 PROJECTS"). Such a syntax aims at satisfying the need to have an expressive and intuitive query language. We furthermore re-use as language keywords the different terms that we have extensively described in previous sections as project information types (e.g. LOC, LICENSE, LANGUAGE, etc.).

Semantics. The semantics of a language defines the conceptual meaning of the sentences (i.e., statements) in that language by stating how they can be logically interpreted. In the case of project search, we reproduce in the DSL the semantics of the data types in natural language. Thus, the term "DESCRIPTION" which refers to the description of a project is also used in the DSL to refer to that information type.

Figure 3 presents the grammar of the DSL in the notation syntax of the Augmented Backus-Naur Form [4]. A query specification (line 1) can be split into 3 parts : (1) the `request` line, where the user specifies the number of projects he is seeking and the information types that must be returned by the engine; (2) some `criteria` statements which describe the criteria that need to be satisfied by the returned results; (3) an optional `preference` part where the requester indicates which results should be returned in priority.

The search criteria may include criteria for picking (*include_filer*) or removing (*exclude_criteria*) candidate projects.

```
query          ::= request ;; (criteria ;;)+ (preference ;;)?
request        ::= SEARCH (number | *) PROJECTS: fields?
fields         ::= attribute attributes*
attribute      ::= NAME | OWNER | ORGANIZATION | URL | HOMEPAGE
                 | LOC | MAIN LANGUAGE | LANGUAGE | CREATION DATE
                 | DESCRIPTION | LABELS | LICENSE | FIRST COMMIT
                 | LAST COMMIT | TESTS | ISSUES | BUGREPS
                 | FEATREQS |NEW FEATREQS | NEW BUGREPS
                 | BUGREPS CLOSED | FEATREQS CLOSED
longcondition  ::= attribute operator numbered WITH subcondition
subcondition   ::= number? qualifier subattr operator expression
shortcondition ::= attribute operator expression
criteria       ::= include-criteria | exclude-criteria
condition      ::= longcondition | shortcondition
preference     ::= PREFER: condition+
include-criteria ::= INCLUDE: condition+
exclude-criteria ::= EXCLUDE: condition+
numbered       ::= number expword? | attribute
attributes     ::= , attribute
word           ::= alphanums (wordpunct alphanums)*
expression     ::= numbered | FAR | NEAR | DIFFERENT | choice
qualifier      ::= AVERAGE | MEDIAN | MAXIMUM | MINIMUM | OTHER
expword        ::= YEARS | MONTHS | DAYS | YEAR | MONTH | DAY
subattr        ::= RESOLUTION | ACTIVITY | TEAM
operator       ::= IS | < | <= | > | >= | = | !=
choice         ::= word (| word)*
alphanums      ::= [a-zA-Z0-9]+
number         ::= [0-9]+
wordpunct      ::= [-_]
```

Fig. 3. Orion Language Grammar

Such group of statements are preceded respectively with the keywords INCLUDE and EXCLUDE, while the preference part is preceded by PREFER. All parts end with the ;; termination sequence. The conditions in the criteria statements express the search criteria using common mathematical relational operators such as '$<$' as well as the semantic operator 'IS'.

We illustrate the expressiveness of the language by specifying search queries for the scenarios described in Section II-A. We provide specifications for possible queries that the researcher Robert, the developer Julia, and the programmer Mark might submit for their search scenarios.

In the first query, the desired projects must have *at least 100 bug reports* and *more than 1000 test cases*. These projects should also have developers from *different communities* (e.g., organizations). To identify relevant projects in this scenario, the project seeker requests the search engine to correlate information for software repository metadata, bug tracking systems, and source code.

Figure 4 describes this specification in Orion DSL. The requested results are limited to 5 instances of projects and must list for each, the name of the project, the organization supporting the project, and the number of bug reports as well as of test case files in the source code. When processing the request, the engine should attempt to return the first 5 projects that are from different organization. The returned projects must only include projects that fit the search conditions described with relational operators.

```
SEARCH 5 PROJECTS: NAME, ORGANIZATION, BUGREPS, TESTS
INCLUDE:
        BUGREPS >= 100
        TESTS > 1000
;;
PREFER:
        ORGANIZATION IS DIFFERENT
;;
```

Fig. 4. Researcher query for relevant datasets

In the second query, the manager could be searching for 4 project that are *largely written* in ANSI C or C++, that are in their early days (*e.g., less than 1 year old*), and are still active (*e.g., last commit happened less than 1 month ago*). Furthermore, the development team for each project must be small (*less than 20 members*), but should include *at least 1 developer* with work experience in a team of *more than 100 developers*.

Figure 5 provides an example specification in Orion DSL for such a search scenario. In this case, the requester tasks the engine to search across all project records and the relevant contributors to identify emerging projects that are adopted by developers having participated in large projects. No particular preferences are specified for this query.

```
SEARCH 4 PROJECTS: NAME, OWNER, MAIN LANGUAGE, LAST COMMIT,
                   NUMBER AUTHORS
INCLUDE:
        MAIN LANGUAGE = ansic|cpp
        LAST COMMIT <= 1 MONTH
        FIRST COMMIT <= 1 YEAR
        NUMBER AUTHORS <= 20 WITH 1 OTHER TEAM > 100
;;
```

Fig. 5. Developer query for relevant projects

In the last query, the programmer is searching for an application dealing, i.e., *tagged*, with "OpenGL" and whose user community is still interested in the project and continues to submit requests for improvements (*the numbers of feature requests and bug reports are each larger than zero*). At the same time, the development team must have *closed more feature requests* and *fixed more bugs* than the numbers of current ones. Finally, the programmer *filters out all licensing schemes (namely GPL v1 and GPL v2)* that conflict with the company policy.

The DSL specification in Figure 6 describes the query for projects that contain candidate code for reuse by a programmer. In this case, the requester imposes search criteria that checks the paradigms and techniques (e.g., webservice, cryptographic algorithms, graphics library etc.) developed or used in the projects. The search also contains an *exclude* block which removes from the results projects that match specific criteria. Finally, this query explicitly suggests a diversification of the results from different project owners. It furthermore sorts the results to return in priority the first 2 projects the longest existence period.

```
SEARCH 2 PROJECTS: NAME, LICENSE, LABELS, SUMMARY, FIRST
                   COMMIT, MAIN LANGUAGE, NEW FEATREQS
                   FEATREQS CLOSED, NEW BUGREPS,
                   BUGREPS CLOSED, LAST COMMIT
INCLUDE:
        LABELS = opengl
        FEATREQS CLOSED > NEW FEATREQS
        BUGREPS CLOSED > NEW BUGREPS
;;
EXCLUDE:
        LICENSE = GPLv1|GPLv2
;;
PREFER:
        OWNER IS DIFFERENT
        CREATION DATE IS FAR
;;
```

Fig. 6. Programmer query for candidate library and code for reuse

The specification examples for the three described scenarios reflect the need for :

- a *knowledge database* which contains a sizeable corpus of

software projects containing a variety of information types that are linked to form project entities which can be queried.

➽ a *dedicated engine* for translating the Orion specification in low-level requests that could be executed by the underlying knowledge database.

B. A Knowledge Database

The Orion integrated search engine is built atop a knowledge database that is constructed in two steps: Firstly, we collect software project data from around the web; Secondly, we infer the links among data from various sources, merge the data, and expose them for queries.

1) Harvesting the web: The World Wide Web has endless amount of information on software development projects. With the momentum on open-source development, a number of platforms have emerged to offer project hosting services. Sourceforge[5] and Google Code[6] are popular examples of such platforms that mainly are built around source code repositories while providing other tools such as bug tracking systems. Recently, the concept of *social coding* has lead to the success of a new kind of development platforms with developer-friendly interfaces [23]. Instances of this concept, including Atlassian Bitbucket[7] and GitHub[8], have increasingly outperformed pioneering platforms. For example, in June 2012, after only 4 years of existence, GitHub is hosting over 3,000,000 repositories, while Sourceforge, launched 13 years ago, contains only about 360,000 projects. Nevertheless, traditional hosting platforms are still used for many projects, and provide various useful features, including extensive project labelling, that are exploited by developers.

Strong organizations, such as the Apache Software Foundation[9] or the Linux community[10], host the development of their projects on their own portals where large amounts of data are available for download. Nonetheless, automatically identifying the homepages of these projects on the web and building tools to parse these web pages to extract project information is challenging. Fortunately, the popularity of hosting platforms and their practical aspects have incited many project managers to register their projects and provide mirrors on GitHub-like platforms, making them easily identifiable.

In this work, we use data collected from GitHub to build the main corpus of software projects for our database. Nevertheless, given a project, its information can be incomplete in GitHub. We therefore augment our data with information extracted from other hosting platforms and project development portals to enrich the amount of information collected for many projects.

a) GitHub - a software project corpus

A decade ago, the task of identifying and collecting a huge number of projects would have been extremingly tedious, due to the scarcity of repositories containing an inventory of projects in development. But, today there are rich assets available that can be harnessed for automatic project information harvesting. Information integration on these platforms is however lacking on such assets, a gap that Orion aims at filling.

GitHub is very convenient for repository mining as it provides extensive REST API[11] for accessing its internal data stores. We have used the API to retrieve general information 100,000 repositories hosted by GitHub. Fig. 7 summarizes the distribution of the GitHub projects in different segments of number of LOC. Over 70% of projects contain more than 1,000 LOC. Around 35% of projects include more than 5,000 projects while more than 20% contain more than 10,000 LOC. Finally, over 600 projects contain more than 1,000,000 LOC. This distribution suggests that a significant number of the projects in the dataset are actually real-world projects.

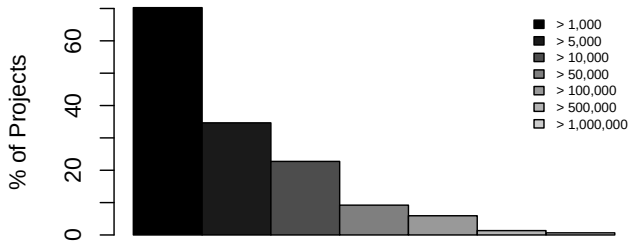

Fig. 7. Distribution of projects in the datasets in terms of total Lines of code

The information collected from GitHub for each project includes a description of the project, the organization/company developing the project, the GitHub user who owns the project, the project homepage, and its name and URL. GitHub also implements bug trackers for each repository and exposes information from these trackers through its APIs. Fundamentally, GitHub bases its operation on git[12], a distributed revision control and source code management system, which keeps track of all contributors by differentiating between revision authors and committers—an important feature in a hierarchical development scheme, such as in Linux, where contributors do not have direct access to the mainline repository.

Nonetheless, the software project corpus thus constituted presents two limitations: (1) it does not contain data on all possible information types for all selected projects. GitHub is indeed developer-oriented and its features may not favour a categorization style where users could identify and select project following the content. Other platforms, which are more user-oriented, may provide better facilities for this particular setting. (2) For some of the project data that are available in hosting platforms, further steps, including downloading raw data, processing and counting, are required to expose relevant information types.

b) Reaching out to other platforms

Aside from projects whose development essentially occurs on GitHub, many others are simply mirrored on GitHub to benefit from broader community exposure. For such projects, the corresponding GitHub repositories lack essential information. For example, the description of the project is often summarized (e.g., "Mirror of Apache CouchDB"[13]) and bug trackers are often disabled as the development occurs outside of GitHub.

[5] sourceforge.net [6] code.google.com [7] bitbucket.org [8] github.com
[9] apache.org [10] kernel.org

[11] api.github.com [12] git-scm.com/ [13] github.com/apache/couchdb

As GitHub is meant for collaborative development, single-man projects for delivering small utilities are often hosted on other platforms such as Google Code and Freecode[14]. Google Code furthermore provides extensive tagging facilities to developers for labelling and categorizing their projects. These tags can be relied upon to improve query results. Due to these considerations, we also collect data for 50,000 projects from Google Code and for 35,000 projects from Freecode. All these project sources provide common information types, such as name and source code, which are already available in the corpus in the case of projects that are also hosted in GitHub. In such cases, the corpus is augmented with information types that are not available in GitHub. However, when a project is only available from one source, the queryable information types are restricted to those available in the dataset. Nevertheless, common queries, including two among the three search scenarios illustrated in this paper, require information types that are common to all platforms.

c) Crawling the web and beyond

While popular software hosting platforms provide bug tracking systems with common features, most of them cannot link bug reports with patches submitted in software revision control systems. GitHub provides a *pull request* feature that can be harnessed to infer those links in cases where the requests are actually accepted by the repository owner. However, the results are often limited to the small set of patches by developers who have no direct access to the main repository. To compensate this deficiency, JIRA, a commercial bug/issue tracking system, provides add-ons for connecting issues to revision control systems and is used by hundreds of projects from the Apache Software Foundation[15]. To collect *bug links*, we crawl the web based on the specific URL format of JIRA.

Finally, we note that other important information for the success of queries, as illustrated by our motivating search scenarios, are not apparent on the hosting platforms. These information include the number of lines of code which can only be inferred after an offline processing of source code. The real distribution of programming languages used in the projects also requires a refined investigation of the source code of each project to differentiate the main language from appearing languages. The source code tree structure from which we deduce e.g., the extent of test cases is also not available directly from project page. Instead, we download every source code repository ourselves in order to explore these properties.

2) Constructing the knowledge database: For the construction of our prototype knowledge database, we have harvested from the collected software projects different types of information and artifacts. This step has required processing project description data, project labels data, code commits, issue/bug reports, source code (for computing the LOC in all languages, and surveying the source tree structrues), etc..

In cases where we merge data from several hosting platforms, since project names are unreliable, we infer the correlation among datasets based on the project's URLs. For example, GitHub repository API provides a *homepage* field where developers can indicate e.g., the Google Code URL from which the project is mirrored. Though we already support a dozen information types[16], extending the database to include additional information should be readily possible.

Finally, the knowledge database contain raw information to ease the maintenance of the update of the knowledge database. Search scnearios that request mean values or other combined information are handled on-the-fly by the underlying engine using virtual tables, such as SQL views. Resource-consuming tasks that were performed to fill the database can now be performed on a small scale for each evolved project. For example, the change of source lines of code can directly be inferred from the new commits without any need for re-computation in the entire source code.

After 1 week of continuous data collection, we have processed about 1.5 terabytes of raw data to fill our MySQL database with 2 gigabytes of harvested information data. To ensure query performance, we have referred to the best practices in MySQL database tuning, and created lookup indices in all tables.

C. Querying with Orion

The Orion search engine takes as input a high-level specification that is checked for correctness and translated into a low-level database request for querying the knowledge database.

correctness checks: The Orion engine first ensures that the specification respects the grammar of the DSL, then performs consistency checks to ensure that the specification does not contain contradictory search criteria.

DB request generation: In the current implementation of the database, the specifications are directly translated to generate appropriate SQL statements which look up multiple tables to serve search requests. Each attribute of the Orion DSL representing an information type that is mapped to a specific table, the Orion compiler iteratively joins the relevant tables that are necessary to answer the requests. A few informations types such as the url are used as keys in the predicates of the JOIN statements. User search preferences are also translated into `GROUP BY` or `ORDER BY` statements depending on the specification. Advanced queries that lead to nested SQL statements are also supported.

Figure 8 provides an illustrative example of the generation outcome for an Orion basic search specification. The specification outlined in Figure 4 was previously used to request researcher datasets in Section II-A where the query relates to information on project organization, on the reported bugs, and on the test cases in the source code. The generated SQL request, in Figure 8, shows that 3 tables (*repos_info, issues, test_cases*), corresponding to different information sources, are joined to serve the request. The use of the Orion DSL achieves two aims: (1) firstly, the Orion DSL abstracts the internal structures of the knowledge database which can become challenging to comprehend; (2) secondly, the DSL allows users who may ignore the underlying requirements of database management systems to easily tune the search criteria, sort the output, and limit the extent of output processing. In our case, for common search scenarios, querying the knowledge database does not require any knowledge on SQL-based relational databases.

[14] freecode.com [15] issues.apache.org

[16] Further descriptions can be found on the project page

Nonetheless, project seekers that are knowledgeable in SQL, can survey the database schemas and construct their own SQL requests to query the knowledge database.

```
SELECT DISTINCT engine.repos_info.name AS 'NAME',
  engine.repos_info.organization AS 'ORGANIZATION',
  engine.issues.number_bugs AS 'BUGREPS',
  engine.test_cases.number_tests AS 'TESTS'
  FROM engine.repos_info
  JOIN engine.issues ON engine.issues.url=engine.repos_info.url
  JOIN engine.test_cases ON engine.test_cases.url=engine.repos_info.url
  WHERE (engine.issues.number_bugs >= 100
      AND engine.test_cases.number_tests > 1000 )
  GROUP BY engine.repos_info.organization
  LIMIT 5
```

Fig. 8. SQL request generated by the Orion compiler for research query (see Orion query in Fig. 4)

Table I shows the result of the researcher query (cf. Fig. 4) that were returned by the Orion search engine.

TABLE I. QUERY RESULTS OF RESEARCH DATASETS

NAME	Organization	# BUGREPS	# TESTS
rails	rails	6760	1318
node	joyent	3491	2709
jboss-as	jbossas	2541	3448
maqetta	maqetta	2596	3137
hiphop-php	facebook	518	5093

Table II describes the output of the developer request (cf. Fig. 5). The user submitted an advanced query whose execution ran across all records of projects and their contributors to identify small-sized projects where some of the developers are also involved in much larger teams.

TABLE II. QUERY RESULTS OF DEVELOPER RELEVANT PROJECTS

NAME	OWNER	MAIN LANGUAGE	LAST COMMIT	NUMBER AUTHORS
PIC	7Robot	ansic	2012-06-03	14
aery32	aery32	ansic	2012-06-09	6
Unuk	Allanis	ansic	2012-06-11	9
Vemulator	apagel	ansic	2012-05-24	6
EpilepsyViewer	Atamai	cpp	2012-06-04	4

The output provided following the specification of the requester does not make apparent the complex combination (cf. Fig. 9) of data sources that was required to resolve the query.[17]

```
SELECT DISTINCT FROM (SELECT t.name AS 'NAME',
    t.owner AS 'OWNER', t.language AS 'MAIN LANGUAGE',
    t.last_commit AS 'LAST COMMIT', nb_authors AS 'NUMBER AUTHORS'
    FROM (SELECT DISTINCT t1.language, t1.last_commit FROM
        (SELECT engine.main_sloc.language, last_commit FROM
            engine.repos_info JOIN engine.projects_activity
            ON engine.projects_activity.url=engine.repos_info.url
            JOIN engine.main_sloc
            ON engine.repos_info.url=engine.main_sloc.url
            WHERE (engine.main_sloc.language='ansic'
                OR engine.relevant_sloc.language='cpp')
            AND last_commit > (NOW() - INTERVAL 1 MONTH)
            AND first_commit > (NOW() - INTERVAL 1 YEAR))t1
        JOIN engine.developers_table
        ON t1.url=engine.developers_tables.url)t
    JOIN engine.developers_in_projects_ordered
    ON engine.developers_in_projects_ordered.author=t.author
    WHERE nb_authors>100
    AND t.chosen_url!= engine.developers_in_projects_ordered.url)t0
    JOIN engine.project_developers_count
    ON engine.project_developers_count.url=t.chosen_url
    WHERE nb_authors<20 limit 4
```

Fig. 9. SQL request corresponding to a relatively-advanced search query (cf. Orion query in Fig. 5)

[17] Due to space limitations, we leave detailed descriptions on the project page

In Table III we show the results of the programmer request. The results are comprised of projects with the "opengl" label with code licenses that are neither GPLv1 nor GPLv2. The query was answered by aggregating project data from GitHub, Google code and source code raw data. The engine exploited the license annotations and the tags provided in Google code by the developer teams. The generated SQL request for this query involves the joining of several tables and an extensive checking of multiple conditions.

IV. ASSESSMENT

In this section, we conduct an evaluation of our approach and of the prototype implementation of Orion. To do so, we perform a user study using the software projects used to build the Orion knowledge database. We explore two aspects: (1) *need*, assessing the opportunity of the Orion approach compared to traditional search engines, and (2) *productivity*, assessing users satisfaction with the results provided by Orion, in terms of query delay and result conformance. We study these aspects through the following research questions:

RQ1. *Need:* Is Orion needed despite the existence of numerous search engines and project hosting platforms?

RQ2. *Productivity:* Does Orion improve productivity, i.e. do project seekers find the results quickly and accurately?

We perform our measurements based on the attitudes of service requesters towards it. To this end, we rely on a psychometric scale, namely the Likert-type scale [18], which specifies the degree if a user agrees or disagrees with a particular issue. The respondents are asked to indicate their degree of agreement by choosing one of five response categories (i.e., we use a 5-point Likert scale). The end-points of a Likert scale are typically "strongly disagree" and "strongly agree". However, in each case of our questionnaires, we have provided further description for the response categories.

The respondents pool was constituted by 13 individuals with different backgrounds : 2 of the respondents have no background on computer programming; 4 respondents are master students; 7 PhD students took part to the study. We consider 10 different search scenarios[18], including the three scenarios that are referred to in previous sections to illustrate researcher, developer and programmer queries.

Execution latency. First we give an overview of of the time required by Orion to answer common queries such as those that were used throughout the paper to illustrate the capabilities of Orion. We have clocked the performance of the Orion search engine when resolving user requests: from the input of Orion specification to the return of search results. For this experiment, the Orion DSL compiler generates SQL requests with the `SQL_NO_CACHE` option to eliminate interference from MySQL cache lookup. We have then recorded query latencies of 0.43 seconds, of 0.54 seconds and of 5.58 seconds respectively for researcher query (cf. results in Table I), developer query (cf. results in Table II) and programmer query(cf. results in Table III).

A. Need

In the absence of Orion, project seekers may still go on the hunt, by themselves, on the world wide web. The

[18] The entire study is available on the project page

TABLE III. QUERY RESULTS OF PROGRAMMER-SOUGHT LIBRARIES

NAME	LICENSE	SUMMARY	LABELS	FEATREQS CLOSED	BUGREPS CLOSED	NEW FEATREQS FEATREQS	NEW BUGREPS	FIRST COMMIT	LAST COMMIT	MAIN LANGUAGE
angleproject	New BSD License	ANGLE: Almost Native Graphics Layer Engine	d3d, google, graphics, html5, OpenGL, OpenGLES, webgl	15	182	8	27	2010-03-03	2012-04-04	cpp
skia	Other Open Source	2D Graphics Library	2D,C,cplusplus,CrossPlatform [...] OpenGL,PDF,Perspective,Vector	10	166	12	87	2006-09-20	2011-09-16	cpp

main instruments at their disposal are the numerous traditional search engines, including Google, Bing, Baidu, etc., the various project hosting platforms, and developer forum portals.

The Likert-type question that we ask in this assessment is : *Could you find the desired projects?* Respondents were asked to answer this question using Orion or traditional search engines.

Respondents were given 1 hour to fill the questionnaire for each of the scenarios and for each of the two methods. Table IV describes the meaning of the different scales in the questionnaires.

TABLE IV. LIKERT SCALE RESPONSE CATEGORIES FOR COMPARING ORION WITH TRADITIONAL SEARCH ENGINES

Scale	Response category
1	I gave up
2	I could not find any
3	I found some but I am not sure
4	I have found some
5	I have found them all

We compare the user experience when relying on Orion and when left with traditional search engines. Figure 10 shows user responses in boxplots created with the R statistical analysis tool. The greyed boxes are delimited by the lines indicating the LOWER QUARTILE, i.e., 25% of the data points are below this line, and the UPPER QUARTILE (25% of the data points are above this line). The bold line in the box plots indicate the MEDIAN (the middle of the dataset). The isolated points (small circles) below the Orion box represent outliers. The horizontal line above the "traditional search engines" box indicate the MAXIMUM (the greatest value). With Orion, users have found some or all the projects requested (scales 4 and 5) while, on average, they could not find any (scale 2) using traditional search engines.

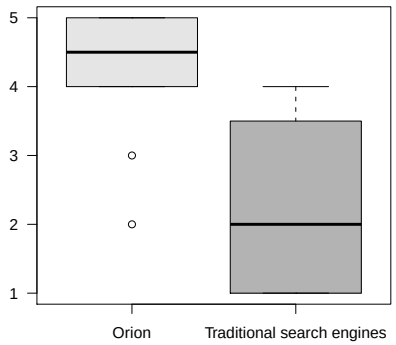

Fig. 10. Assessing the need – Comparison of search results

We furthermore use the Mann-Whitney-Wilcoxon test [16], a non-parametric test, to assess the statistical difference in the distributions of user responses for Orion and traditional search engines. We find that the distributions of user responses are different at a significant level of 0.001, strengthening our findings that Orion was very well assessed by users.

Orion outperforms traditional search engines in finding relevant software projects.

B. Productivity

One of the benefits of DSLs is to enhance productivity [14]. In the last user study we investigate how Orion improves the quest for relevant software projects. This study complements the study on the usability of Orion by assessing the search output for user-imagined queries through the question *Are you satisfied with the returned projects?*.

Figure 11 draws the boxplot representing the distribution of user's satisfaction towards the results returned by Orion. In general, users have found relevant projects in response to their queries. The average being between 4 and 5, the study suggests that the results returned by Orion appear to correspond to the search intents. All responders allocated a maximum of 1 hour slot to query and check the results. We note that this amount of time is reasonable as in the first study, for many scenarios with traditional search engines, users gave up or did not find any project at the end of the allocated time.

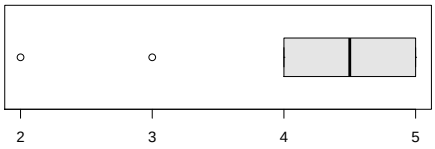

Fig. 11. Assessing the productivity of Orion

Project seekers find, with Orion, accurate results in a relatively limited period of time.

V. RELATED WORK

We discuss in the following a number of studies related to the collection of project data, to project and code search, and to semantic search implementations.

Collection of software project data. In [21], Nagappan has reported on the discussions of a working group that describes the opportunities and challenges in using open source software repositories for empirical studies. In the past years, significant effort has been spent into collecting, curating, and analyzing data from open source projects around the world. The FLOSSMole project[19], initially collecting Sourceforge projects, includes datasets from various sources in various formats. The Flossmetrics [7] and Sourcerer [1] projects furthermore provide statistics on their collected data. These global statistics are not suitable for selecting a subset or identifying a unique project based on desired properties.

Bird *et al.* have provided insightful discussions on the opportunities and challenges of mining Git repositories [2], while Dabbish *et al.* have presented a study that investigates the impact of transparency in GitHub through a series of interviews [5]. More recently, the GHTorrent project aims at bringing GitHub's rich product and process data to the hands of research community [9]. Unfortunately, they do not improve this collection for the many projects whose real development

[19] flossmole.org

activities occur outside of GitHub. The FRASR framework for analyzing software repositories has mostly focused on addressing the challenges for data collection and curation [22]. They do not exploit the large datasets in GitHub and its clean APIs. The TA-RE project aims to facilitate sharing of benchmark datasets among researchers [15], through an exchange language.

Overall, our work improves over previous efforts by merging various artifacts from various sources, and provides the capabilities for executing complex queries that can yield more relevant search results for different types of project seekers.

The SeCold [13] platform links data sources and creates *facts*, which are analysis results published by developers who gathered them with their own tools. Similar to SeCold, Orion integrates and links a variety of information. The BOA [6] language and infrastructure also aim at improving MSR studies by reducing the workload and increasing the efficiency in computing statistics (e.g. average size of commits) across repositories. Orion, on the other hand, is targeted at the selection of specific projects which for search purposes are considered as entities with rankable properties. These two approaches thus target different usage scenarios. Additionally, different from BOA, Orion *integrates* different sources of information including: GitHub, Google Code, JIRA, etc.

Projects, applications and code search. Project hosting platforms such as GitHub offer search facilities in their user interfaces and through their APIs. The capabilities of these search engines are however limited to a few information types (e.g., search by language), and do not allow advanced queries across on multiple criteria. Finally, search results are returned as unstructured text [20]. CodeFinder [11], CodeGenie [17], CodeBroker [24], Examplar [10] and PortFolio [20] identify and return relevant projects or functions by mining source code repositories. Orion on the hand is focused at searching for project entities based on different search criteria for diverse development artifacts.

Similar to our project discovery infrastructure, the *Ohloh*[20] graphical interface, provides users with updated ranking of projects (e.g., popular projects, most active projects, etc.). Orion is more powerful as it considers many more development artifacts. Thus, a much wider variety of search queries is supported.

Semantic search. Real world implementations of semantic search have been deployed. Example such as Wolfram Alpha, focus on a restricted domains for more accuracy. The Semantic MediaWiki[21] framework includes a simple query language for semantic search, so that users can directly request certain information from the wiki database. Orion follows the same approach by providing a user-friendly language for specifying project search requests.

VI. Conclusion & Future work

In this paper, we have introduced Orion, a unified platform for searching relevant software projects leveraging various data sources including source code and revision control information, bug reports and developer activity. We have downloaded and processed software development artefacts for tens of thousands of software projects on popular hosting platforms and crawled development artifacts data on the web to build our prototype knowledge database. Orion enables the execution of various complex queries that are not supported by traditional web search engines. Orion furthermore comes with a language target at project search.

We have demonstrated the need for Orion as well as its productivity by conducting a user-study. The results indeed show with strong statistical significance that users find relevant software projects faster and more accurately with Orion. This work is being expanded in various directions for (1) enriching the knowledge database with more software projects, (2) exposing more information types for queries and (3) improving the engine towards the implementation of a transparent semantic query.

Availability: Materials for this paper, including information on the collected projects, the database, the user study, etc. can be found at http://momentum.labri.fr/orion.

[20] http://ohloh.net [21] semantic-mediawiki.org

References

[1] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An internet-scale software repository," in *SUITE*, 2009.

[2] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. Germán, and P. T. Devanbu, "The promises and perils of mining git," in *MSR*, 2009.

[3] T. Cheng and K. C.-C. Chang, "Entity search engine: Towards agile best-effort information integration over the web," in *CIDR*, 2007.

[4] D. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," RFC 5234 (Standard), IETF, Jan. 2008.

[5] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *CSCW*, 2012.

[6] R. Dyer, H. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *ICSE*, 2013.

[7] J. M. Gonzalez-Barahona, G. Robles, and S. Dueñas, "Collecting data about floss development: the flossmetrics experience," in *FLOSS*, 2010.

[8] G. Gousios and D. Spinellis, "A platform for software engineering research," in *MSR*, 2009.

[9] ——, "Ghtorrent: Github's data from a firehose," in *MSR*, 2012.

[10] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *ICSE*, 2010.

[11] S. Henninger, "Supporting the construction and evolution of component repositories," in *ICSE*, 1996.

[12] J. Howison, M. Conklin, and K. Crowston, "FLOSSMole: a collaborative repository for FLOSS research data and analyses," *IJITWE*, vol. 1, no. 3, Jul. 2006.

[13] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, "A linked data platform for mining software repositories," in *MSR*, 2012.

[14] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton, "A software engineering experiment in software component generation," in *ICSE*, 1996.

[15] S. Kim, T. Zimmermann, M. Kim, A. Hassan, A. Mockus, T. Girba, M. Pinzger, E. J. Whitehead, Jr., and A. Zeller, "TA-RE: An exchange language for mining software repositories," in *MSR*, 2006.

[16] W. H. Kruskal, "Historical notes on the wilcoxon unpaired two-sample test," *Journal of the American Statistical Association*, vol. 279, no. 52, Sep. 1957.

[17] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes, "Codegenie: using test-cases to search and reuse source code," in *ASE*, 2007.

[18] R. Likert, "A technique for the measurement of attitudes." *Archives of Psychology*, vol. 22, no. 140, 1932.

[19] J. R. Mashey, "Languages, levels, libraries, and longevity," *Queue*, vol. 2, no. 9, Dec. 2004.

[20] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *ICSE*, 2011.

[21] N. Nagappan, "Potential of open source systems as project repositories for empirical studies *working group results*," in *Empirical Software Engineering Issues*, 2006.

[22] W. Poncin, A. Serebrenik, and M. van den Brand, "Process mining software repositories," in *CSMR*, 2011.

[23] M.-A. D. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *FoSER*, 2010.

[24] Y. Ye and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," in *ICSE*, 2002.