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*—What projects contain more than 10,000 lines of code developed by less than 10 people and are still actively maintained with a high bug-fixing rate? To address the challenges for answering such enquiries, we develop an integrated search engine architecture that combines information from different types of software repositories from multiple sources. Our search engine facilitates the construction and execution of complex search queries using a uniform interface that transparently correlates 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 various software development artifacts. Using scenarios, we illustrate the benefits of such a search engine for different kinds of project seekers.

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 [3]–[5]. 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. These challenges also apply to developers who can hardly determine which projects would benefit the most from their expertise while being a "friendly" development environment for them, and to users who often experience frustration in their quest for the "right" software to use.

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], [5], or aimed at regularly delivering snapshots and statistics on monitored projects [6]. 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.

The main contributions of this paper are as follows:

- We discuss the opportunities and challenges in the context of project search by exploring a few search scenarios.
- We propose the Orion search engine for empowering users, developers and researchers in their quests.
- We highlight 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 projects.

Section II motivates this work and outlines the challenges

The work was done while the author was visiting Singapore Management University

978-0-7695-5007-7/13 $26.00 © 2013 IEEE
DOI 10.1109/ICECCS.2013.42

that we face in the design of a project search engine. Section III and IV discuss our approach and related work. Finally, Section V summarizes the status and future work.

II. Motivations

Thanks to the momentum of open source philosophy, a wide range of project development artifacts are available on project homepages and on hosting platforms, creating 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. We now describe complex and challenging search scenarios.

A. Search Scenarios

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.

Unfortunately, this search is challenging using the traditional approach. 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.

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.

As in the previous scenario, 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 his goals.

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) Dispersed Information: Information on development artifacts of software projects are stored in heterogeneous repositories which are often located at various decorrelated sites and 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 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* [2]. 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.

III. Orion

To address the challenges for finding relevant software projects, two essential requirements must be met, namely (1) information integration to limit the impact of the heterogeneity of software repositories as well as their dispersion in various locations and (2) the possibility for an effective search through an expressive query system. The design of the Orion search engine approach, illustrated in Figure 1, is motivated by these requirements. 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.

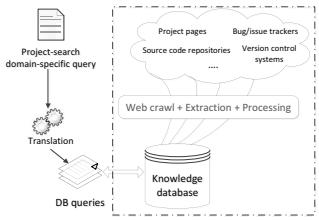

Fig. 1. The Orion approach

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

A software **project entity** includes a variety of information types including *project metadata* such as the project name, its owner, its description and a set of labels defining it, *source code information* such as the used programming languages, the number of lines of code and the presence of test cases, *bug reports and feature requests*, *development context* information such as project creation date, different code commit dates, size of development team and developers. All these information types must be captured by the DSL and should be readily extractable from the knowledge database.

A. A Declarative Query Language

Expressive and intuitive queries are essential for improving the user-experience in the quest of software projects. The Orion DSL[3] was built to capture the characteristics of software projects, the variety of search criteria, and the readiness at which users can navigate across project data. This DSL is close to *human natural* language and uses common English keywords and expressions (e.g., "SEARCH 3 PROJECTS").

We illustrate the expressiveness of the language by providing a specification for a possible query that programmer Mark (cf. Section II-A) might submit for his search scenario. In this 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 *honored 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 his company policy.

The DSL specification in Figure 2 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 with 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. 2. Programmer query for candidate library and code for reuse

This specification example highlights 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 and for 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[4] and Google Code[5] and the GitHub[6] *social coding* site are popular examples of such platforms that mainly are built around source code repositories while providing other tools such as bug tracking systems. Strong organizations, such as the Apache Software Foundation[7] or the Linux community[8], host the development of their projects on their own portals where large amounts of data are available for download.

a) GitHub - a software project corpus In this work, we use data collected from GitHub to build the main corpus of software projects for our database. This platform is very convenient for repository mining as it provides an extensive REST API[9] for accessing its internal data stores. We have used the API to retrieve general information on 100,000 repositories hosted by GitHub. 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.

b) Reaching out to other platforms and beyond the web 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[10]. 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.

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.

[3] Due to space constraints, further information on the DSL is only available at http://momentum.labri.fr/orion

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

TABLE I
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

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.

In total, our knowledge database supports a dozen information types and is extensible to include additional types. 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.

In Table I 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.

Table II shows the result of the researcher query that were returned by the Orion search engine. In this case, the specification[11] requested 5 projects with *at least 100 bug reports and 1000 test cases* from different development communities. To identify those relevant projects, Orion correlated information from repository metadata, bug tracking systems, and source code.

TABLE II
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

IV. RELATED WORK

In [9], 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[12], Flossmetrics [3] and Sourcerer [1] projects collect data and/or provide statistics on their collected data, but are not suitable for selecting a subset or identifying a unique project based on desired properties.

The GHTorrent project aims at bringing GitHub's rich product and process data to the hands of research community [5]. Unfortunately, contrary to Orion, they do not improve this collection for the many projects whose real development activities occur outside of GitHub. The FRASR framework for analyzing software repositories has mostly focused on addressing the challenges for data collection and curation [10]. 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 [7], through an exchange language.

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 [8].

V. CURRENT STATUS & FUTURE WORK

In this paper, we have introduced Orion, a unified platform for searching relevant software projects. 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 targeted at project search.

We continue to collect more project data into our knowledge database. We are currently assessing the Orion approach through a large user study to explore four aspects: (1) the actual *need* for Orion, i.e., how does it compare with traditional search engines?, (2) its *usability/expressiveness* to evaluate the intuitiveness and capabilities of the DSL, and (3) its *productivity* in terms of the satisfaction of users towards the results yielded by the search engine.

[11] More information at http://momentum.labri.fr/orion [12] flossmole.org

REFERENCES

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

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

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

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

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

[6] J. Howison, M. Conklin, and K. Crowston, "FLOSSMole: a collaborative repository for FLOSS research data and analyses," *IJITWE*, Jul. 2006.

[7] 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.

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

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

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