Perceval: Software Project Data at Your Will

Santiago Dueñas
Bitergia
sduenas@bitergia.com

Valerio Cosentino
Bitergia
valcos@bitergia.com

Gregorio Robles
Universidad Rey Juan Carlos
grex@gsyc.urjc.es

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos
jgb@gsyc.urjc.es

ABSTRACT

Software development projects, in particular open source ones, heavily rely on the use of tools to support, coordinate and promote development activities. Despite their paramount value, they contribute to fragment the project data, thus challenging practitioners and researchers willing to derive insightful analytics about software projects. In this demo we present Perceval, a loyal helper able to perform automatic and incremental data gathering from almost any tool related with contributing to open source development, among others, source code management, issue tracking systems, mailing lists, forums, and social media. Perceval is an industry strong free software tool that has been widely used in Bitergia, a company devoted to offer commercial software analytics of software projects. It hides the technical complexities related to data acquisition and eases the definition of analytics. A video showcasing the main features of Perceval can be found at https://youtu.be/eH1sYF0Hdc8.

KEYWORDS

Software mining, empirical software engineering, open source software, software development, software analytics

ACM Reference Format:
Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M. Gonzalez-Barahona. 2018. Perceval: Software Project Data at Your Will. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183475

1 INTRODUCTION

In the last years, platforms like GitHub, StackOverflow, Slack or Meetup have become important tools to support, coordinate and promote the daily activities around software. This is specially true for open source projects, which rely heavily on distributed and collaborative development [5, 18].

Beyond being successfully and increasingly adopted by both end-users and development teams [23], these tools contain relevant information about the history of a software project, thus they can be exploited by practitioners and researchers to understand, predict, and improve specific aspects of the projects [13, 15].

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3183475

However, accessing and gathering this data is often a time-consuming and an error-prone task, that entails many considerations and technical expertise [1, 12, 16]. It may require to understand how to obtain an OAuth [11] token (e.g., StackExchange, GitHub) or prepare storage to download the data (e.g., Git repositories, mailing list archives); when dealing with development support tools that expose their data via APIs, special attention has to be paid to the terms of service (e.g., an excessive number of requests could lead to temporary or permanent bans); recovery solutions to tackle connection problems when fetching remote data should also taken into account; storing the data already received and retrying failed API calls may speed up the overall gathering process and reduce the risk of corrupted data. Nonetheless, even if these problems are known, many scholars and practitioners tend to re-invent the wheel by retrieving the data themselves with *ad-hoc* scripts.

In this paper, we present `Perceval`, a tool that simplifies the collection of project data by covering more than 20 well-known tools and platforms related to contributing to open source development, thus enabling the definition of software analytics. It rebuilts and extends previous tools that have been widely used in academia, such as `CVSAnalY` [22], `MailingListStats` and `Bicho` [20][1].

Perceval is an industry-strength tool, that (i) allows to retrieve data from multiple sources in an easy and consistent way, (ii) offers the results in a flexible JSON format, and (iii) gives the possibility to connect the results with analysis and/or visualization tools. Furthermore, it is easy to extend, allows cross-cutting analysis and provides incremental support (useful when analyzing large software projects).

Several tools aimed at collecting project data have been proposed in the past decade by other research groups. However, they all fall short when compared with `Perceval`. Most tools generally focus on specific aspects of the project, thus they are not designed to be easily extended, and transversal analysis are restricted to only few data sources [2, 8]. `Gitana` [4] and `Kibble`[2] are similar to `Perceval` in their goals, but are not industrial-strong and cover much less data sources. `GHTorrent` [10] and `BOA` [7] are powerful infrastructures that provide periodical snapshots of project activities, but they are limited only to the GitHub and SourceForge ecosystems.

The remainder of the paper is organized as follows: Section 2 briefly describes the approach underlying `Perceval`. Section 3 shows how `Perceval` works, while Section 4 reports on how the tool has been used and validated, allowing it to be considered a mature tool. Finally, Section 5 concludes the paper.

[1] At the time of writing, Google Scholar offers more than 300 references of academic publications that make use of these tools.
[2] https://kibble.apache.org

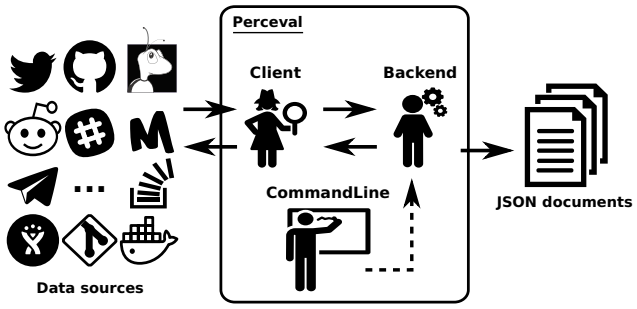

Figure 1: Overview of the approach. The user interacts with the backend through the shell command line, which depending on the data source retrieves with a specific client the data; the output is provided in form of JSON documents.

2 OVERVIEW OF PERCEVAL

Perceval was designed with a principle in mind: *do one thing and do it well*. Its goal is to fetch data from data source repositories efficiently. It does not store nor analyze data, leaving these tasks to other, specific tools. Though it was conceived as a command line tool, it may be used as a Python library as well. It was named after Perceval (Percival or Parsifal in different translations), one of the knights of King Arthur and a member of the Round Table, who after suffering pains and living adventures was able to find the Holy Grail.

Currently, `Perceval` covers the following data sources, which are commonly used to support, coordinate and promote development activities:

- Version control systems: Git
- Source code review systems: Gerrit, GitHub.
- Bugs/Ticketing tools: Bugzilla, GitHub, JIRA, Launchpad, Phabricator, Redmine, GitLab
- Mailing: Hyperkitty, MBox archives, NNTP, Pipermail
- News: RSS, NNTP
- Continous integration: Jenkins
- Instant messaging: Slack, Supybot archives (IRC), Telegram
- Q/A: Askbot, Discourse, StackExchange
- Documentation: Confluence, Mediawiki
- Others: DockerHub, Meetup

A common execution of `Perceval` consists of fetching a collection of homogeneous items from a given data source. For instance, issue reports are the items extracted from Bugzilla and GitHub issues trackers, while commits and code reviews are items obtained from Git and Gerrit repositories. Each item is inflated with item-related information (e.g., comments and authors of a GitHub issue) and metadata useful for debugging (e.g., backend version and timestamp of the execution). The output of the execution is a list of JSON documents (one per item).

The overall view of `Perceval`'s approach is summarized in Figure 1. At its heart there are three components: *Backend*, *Client* and *CommandLine*.

Listing 1: How to install Perceval from pip and source code.

```
# Installation through pip
$ pip3 install perceval
------------------------------------------------------
# Installation from source code
$ git clone https://github.com/grimoirelab/perceval.git
$ pip3 install -r requirements.txt
$ python3 setup.py install
```

2.1 Backend

The *Backend* orchestrates the gathering process for a specific data source and puts in place incremental and caching mechanisms. Backends share common features, such as incrementality and caching, and define also specific ones tailored to the data source they are targeting. For instance, the GitHub backend requires an API token and the names of the repository and owner; instead the StackExchange backend needs an API token plus the tag to filter questions.

2.2 Client

The backend delegates the complexities to query the data source to the *Client*. Similarly to backends, clients share common features such as handling possible connection problems with remote data sources, and define specific ones when needed. For instance, long lists of results fetched from GitHub and StackExchange APIs are delivered in pages (e.g., pagination), thus the corresponding clients have to take care of this technicality.

2.3 CommandLine

The *CommandLine* allows to set up the parameters controlling the features of a given backend. Furthermore, it also provides optional arguments such as *help* and *debug* to list the backend features and enable debug mode execution, respectively.

3 PERCEVAL IN ACTION

This section describes how to install, use and exploit `Perceval`, highlighting its main features.

3.1 Installation

`Perceval` is being developed and tested mainly on GNU/Linux platforms. Thus, it is very likely it will work out of the box on any `Linux`-like (or `Unix`-like) platform, upon providing the right version of `Python` available.

There are several ways for installing `Perceval` on your system: with the `pip` packager manager[3], from a `Docker` image or from the source code. Listing 1 shows how to install `Perceval` from `pip` and source code. Further installation information can be found in the GitHub repository hosting the tool[4].

3.2 Use

Once installed, a `Perceval` backend can be used as a stand-alone program or Python library. We showcase these two types of executions by fetching data from a Git repository[5].

[3] https://pypi.python.org/pypi/pip
[4] https://github.com/grimoirelab/perceval
[5] Additional examples of usage are available at https://github.com/grimoirelab/training/tree/master/perceval

Listing 2: JSON document excerpt of a git commit.

```
{
   "backend_name": "Git",
   "backend_version": "0.3.0",
   "data": {
      "CommitDate": "Tue Aug 18 18:08:27 2015 +0200",
      "commit": "dc78c254e464ff334892e0448a23e4cfbfc637a3",
      "files": [
         {
            "action": "A",
            "added": "10",
            "file": ".gitignore", ...
         }, ...
      ],
      "message": "Initial import", ...
   },
   "origin": "https://github.com/grimoirelab/perceval.git", ...
}
```

Listing 3: Using Perceval as a program.

```
$ perceval git https://github.com/grimoirelab/perceval
> /perceval.test
[2017-11-18 20:32:19,425] - Sir Perceval is on his quest.
[2017-11-18 20:32:19,427] - Fetching commits:
'https://github.com/grimoirelab/perceval' git repository
from 1970-01-01 00:00:00+00:00; all branches
[2017-11-18 20:32:20,798] - Fetch process completed:
798 commits fetched
[2017-11-18 20:32:20,798] - Sir Perceval completed his quest.
```

Git is probably the most popular source code management system nowadays. Is is usually used to track versions of source code files. Transactions on a Git repositories are called *commits*. Each commit is an atomic change to the files in the repository. For each commit, Git maintains data for tracking what changed, and some metadata such as who committed the change, when and which files were affected.

`Perceval` clones the Git repository to analyze, and gets information for all its commits by using the `git log` command under the hoods. It produces a JSON document (a dictionary when using it from Python) for each commit. Listing 2 shows an excerpt of a JSON document produced. As can be seen, the document contains some item-related information (e.g., *files*) plus metadata included by `Perceval` itself (e.g., *backend_name*, *backend_version*).

3.2.1 Stand-alone Program. Using `Perceval` as stand-alone program does not require much effort, but only some basic knowledge of GNU/Linux shell commands. Listing 3 shows how easy it is to fetch commit information from a Git repository. As can be seen, the backend for Git requires the URL where the repository is located (*https://github.com/grimoirelab/perceval.git*), then the JSON documents produced are redirected to the file *perceval.test*. The remaining messages in the listing are prompted to the user during the execution.

One interesting optional argument is *from-date*, which allows to fetch commits from a given date, thus showing an example of how incremental support is easily achieved in `Perceval`.

3.2.2 Python Library. `Perceval`'s functionalities can be embedded in Python scripts. Again, the effort of using `Perceval` is minimum. In this case the user only needs some knowledge of Python scripting.

Listing 4 shows how to use `Perceval` in a script. The *perceval.backends module* is imported at the beginning of the file, then

Listing 4: Using Perceval as a library.

```
#! /usr/bin/env python3

from perceval.backends.core.git import Git

# URL for the git repo to analyze
repo_url = 'http://github.com/grimoirelab/perceval'
# directory for letting Perceval clone the git repo
repo_dir = '/tmp/perceval.git'

# Git object, pointing to repo_url and repo_dir for cloning
repo = Git(uri=repo_url, gitpath=repo_dir)
# fetch all commits and print each author
for commit in repo.fetch():
        print(commit['data']['Author'])
```

the *repo_url* and *repo_dir* variables are set to the URL of the Git repository and the local path where to clone it. These variables are used to initialize an object of the *perceval.backends.git.Git* class. In the last two lines of the script, the commits are retrieved using the method *fetch* and the names of their authors printed. The *fetch* method, which is available in all backends, needs to be tailored to the target data source. Therefore, the Git backend fetches commits, while GitHub and StackExchange ones fetch issues and questions. The *fetch* method optionally accepts a *Datetime* object to gather only those items in the data source modified after a given date. When possible, the filtering of the items relies on the data source functionalities, for instance the GitHub API allows to ask for issues modified after a date. In the other cases, the filtering is implemented in the backend itself.

3.3 Exploitation

The JSON documents obtained by Perceval can be exploited by storing them to a persistent storage, such as an `ElasticSearch` database [9]. The documents can be then accessed via the `ElasticSearch` REST API using built-in modules (e.g., *elasticsearch_dsl*) and simple off-the-shelf tools (e.g., *curl* or Python libraries such as *urllib* or *Requests*). Finally, the data can be processed with common libraries for data analytics like `Pandas` [19] and `R` [24], or visualized by means of `Kibana` dashboards.

3.3.1 Data Analysis (and Prototyping). `Pandas` [19] is one of the most common libraries used in data analytics with Python. It can be very useful when dealing with the output of `Perceval`. `Jupyter Python Notebooks` can be used for data analysis and early prototyping of data visualization [17]. Since the process is completely transparent –`Python Notebooks` show the results and the code needed to produce them–, researchers and practitioners can examine the output and perform changes fast.

3.3.2 Visualization. True interactive dashboards can be produced by relying on the `GrimoireLab` toolset (to which Perceval belongs to). `GrimoireLab` provides Python components which handle the data obtained by `Perceval`, drive its uploading and enrichment to `ElasticSearch`, and produce everything needed to deploy dashboards tailored to software project analytics (see Figure 2).

4 USE IN ACADEMIA AND INDUSTRY

`Perceval` is a very valuable tool for both education and research purposes. Educators can easily use it to teach software engineering approaches for mining data from repositories, since students

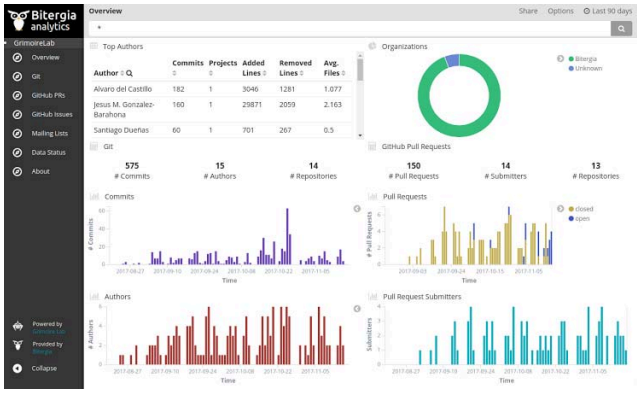

Figure 2: Dashboard produced by Kibana with data gathered with Perceval. An analysis of two different sources, git (commits, authors, top authors and organizations) and GitHub (pull requests, pull requests submitters), is shown.

require only basic knowledge to perform the initial data downloading and formatting. Also researchers can benefit of Perceval to extract software project data for their studies, since it already covers popular data sources and it is easy to extend (i.e., around 300 lines of code are needed to support a new data source). For instance, previous research efforts [3, 6, 14, 21] have used Perceval to analyse activities of Git repositories and developers.

Furthermore, Perceval can be considered a mature tool since it is used in Bitergia to collect data from more than 30,000 development support tools, where Git (19,671), GitHub (4,496) and Bugzilla (2,229) are the most common ones. Thus, Perceval is the basis of the Bitergia infrastructure for builiding dashboards for its clients, such as The Linux Foundation, Red Hat, Wikimedia Foundation or The Document Foundation[6].

The collaboration with these communities has been fundamental, since developers and managers can easily inspect their project data. Thus, it helps to increase trust and transparency of the project and better assess its health.

5 CONCLUSIONS AND FUTURE PLANS

Mining software repositories has been an area of a high academic and practitioner activity in the last years, especially in the open source field.

In this demonstration paper we have presented Perceval, the next generation of already popular tools used in academia and industry. Perceval departs from our previous experience to create and industry-strength tool that is currently under deployment in major professional settings. It has been designed to mine many software development data sources, including the major development support tools, such as versioning systems, bug tracking systems and mailing lists archives. Perceval's output is in form of JSON documents, which allows for flexibility and ease of connection with other tools to perform the analysis and/or visualization of the data.

[6]Bitergia public dashboards can be seen at: https://bitergia.com/products/dashboards/

Perceval offers the automatic and incremental data gathering functionality of the GrimoireLab toolset in which Bitergia is currently working. The aim of GrimoireLab is to provide an open source platform that in addition offers (i) automatic gathered data enrichment, merging duplicated identities, adding additional information about contributors affiliation, calculation delays, geographical data, and (ii) data visualization, allowing filtering by time range, project, repository, contributor, among others.

REFERENCES

[1] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. 2009. The promises and perils of mining git. In *MSR*. 1–10.
[2] Casey Casalnuovo, Yagnik Suchak, Baishakhi Ray, and Cindy Rubio-González. 2017. GitcProc: a tool for processing and classifying GitHub commits. In *ISSTA*. 396–399.
[3] Maëlick Claes, Mika Mäntylä, Miikka Kuutila, and Bram Adams. 2017. Abnormal Working Hours: Effect of Rapid Releases and Implications to Work Content. In *MSR '17*. IEEE Press, 243–247.
[4] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Gitana: a SQL-based Git Repository Inspector. In *ER*. 329–343.
[5] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *CSCW*. 1277–1286.
[6] Premkumar Devanbu, Pallavi Kudigrama, Cindy Rubio-González, and Bogdan Vasilescu. 2017. Timezone and Time-of-day Variance in GitHub Teams: An Empirical Method and Study. In *SWAN '17*. ACM, 19–22.
[7] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *MSR*. 422–431.
[8] Michael Fischer, Martin Pinzger, and Harald Gall. 2003. Populating a release history database from version control and bug tracking systems. In *ICSM 2003*. 23–32.
[9] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. "O'Reilly Media, Inc.".
[10] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's data from a firehose. In *MSR 2012*. 12–21.
[11] Dick Hardt. 2012. The OAuth 2.0 authorization framework. (2012).
[12] Hadi Hemmati, Sarah Nadi, Olga Baysal, Oleksii Kononenko, Wei Wang, Reid Holmes, and Michael W Godfrey. 2013. The MSR cookbook: Mining a decade of research. In *MSR*. 343–352.
[13] Israel Herraiz, Daniel Izquierdo-Cortazar, and Francisco Rivas-Hernández. 2009. Flossmetrics: Free/libre/open source software metrics. In *CSMR'09*. 281–284.
[14] Truong Ho-Quang, Regina Hebig, Gregorio Robles, Michel RV Chaudron, and Miguel Angel Fernandez. 2017. Practices and perceptions of UML use in open source projects. In *ICSE SEIP*. IEEE Press, 203–212.
[15] James Howison, Megan Conklin, and Kevin Crowston. 2006. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *Intl Journal of Information Technology and Web Engineering* 1, 3 (2006), 17–26.
[16] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071.
[17] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, and others. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.
[18] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaíno. 2010. Collaboration tools for global software engineering. *IEEE software* 27, 2 (2010).
[19] Wes McKinney and others. 2010. Data structures for statistical computing in python. In *SciPy*, Vol. 445. 51–56.
[20] Gregorio Robles, Jesús M González-Barahona, Daniel Izquierdo-Cortazar, and Israel Herraiz. 2009. Tools for the study of the usual data sources found in libre software projects. *Intl J of Open Source Software and Processes* 1, 1 (2009), 24–45.
[21] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel RV Chaudron, and Miguel Angel Fernandez. 2017. An extensive dataset of UML models in GitHub. In *MSR'17*. IEEE Press, 519–522.
[22] Gregorio Robles, Stefan Koch, and Jesús M González-Barahona. 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In *2nd Workshop Remote Analysis and Measurement of Softw Systems*. 51–56.
[23] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te Cheng. 2010. The impact of social media on software engineering practices and tools. In *FSE*. ACM, 359–364.
[24] R Core Team. 2000. R language definition. (2000).