Mining Software Engineering Data from GitHub

Georgios Gousios and Diomidis Spinellis

May 23, 2017

GitHub and their data

With 40M repos and 15M users, GitHub is the largest source code archive and one of the largest online collaboration platforms on the planet.

For software engineering researchers, GitHub is valuable because:

- It has an extensive API: api.github.com
- It exposes both process and product data
- All data are highly interconnected

GitHub's API

Contains all data from all public repositories

- *Events*: A real-time endpoint of all things happening on GitHub
- *Entities*: Represent the state of a resource at the time of query
- *Graphs*: Query entities in a depth-first fashion (new!)

Ways of accessing GitHub data

- REST API / GraphQL
- GitHub Archive Collects GitHub events and offers them over BigQuery
- Github on BigQueury The source code of all public GitHub repos along with metadata on BigQuery.
- GHTorrent Collects GitHub events, resolves all entities linked from those and creates a relational view. Data is offered as downloads, online access services or over BigQuery.

The REST API

The REST API allows us to browse entities given a known starting point

```
# Find Microsoft's 5 most starred repos
curl -s "https://api.github.com/orgs/microsoft/repos?per_page=100&page=1" |
jq 'sort_by(.stargazers_count) |.[] | [.name, .stargazers_count] | @csv'|
sed -e 's/^"\(.*\)"$/\1/' |tr -d '\' |
tail -n 5
```

```
## "referencesource",1207
## "cpprestsdk",1585
## "TypeScriptSamples",1825
## "dotnet",8031
## "TypeScript",20824
```

- An API key is needed, 5k reqs/hour
- Always use in combination with `per_page=100`
- Too fine-grained for many MSR tasks

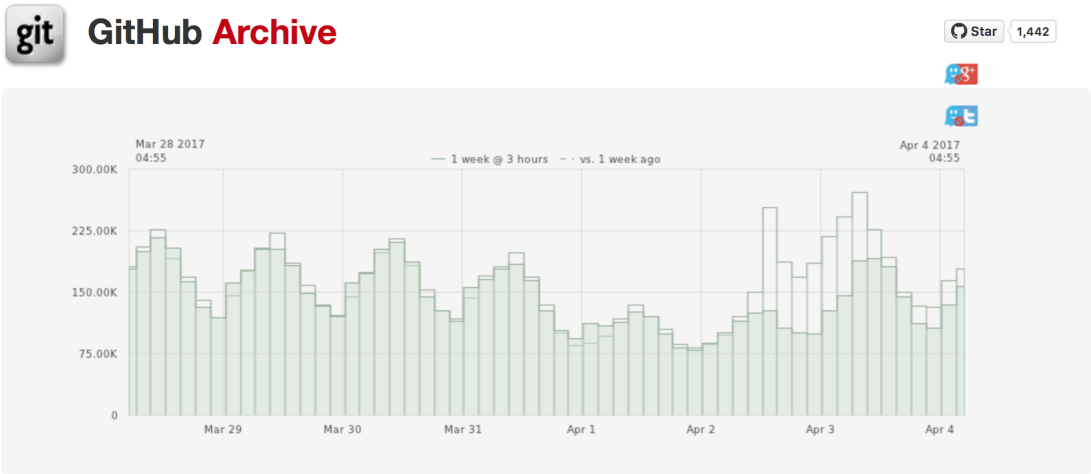

Figure 1: GitHub Archive

GraphQL API

GraphQL API allows us to query entities and their dependents in one go

```
{
  user(login: "dhh") {
    name
    location
    organizations(first:10) {
      nodes {
        name
        repositories(first: 20) {
          nodes {
            name
          }
        }
      }
    }
  }
}
```

https://developer.github.com/v4/explorer/

GitHub Archive

Collects all GitHub **events** since late 2011, allows querying over Google BigQuery.

GitHub data on BigQuery

An updating snapshot of both code + metadata for 2.5M repos.

What are the most popular testing frameworks for Python?

```
SELECT lib, count(*) AS count
FROM (
  SELECT REGEXP_EXTRACT(versions, r"(\w*)") as lib
  from (
    SELECT first(split(c.content, '\n')) as versions
    FROM (
        SELECT *
        FROM [bigquery-public-data:github_repos.files]
        WHERE path contains 'requirements.txt'
      ) AS f
      JOIN
         [bigquery-public-data:github_repos.contents] c on f.id = c.id
    ) AS v
  WHERE REGEXP_MATCH(versions, "[>=]=[0-9]*")
) AS l
GROUP BY lib
ORDER BY count DESC
```

GitHub data on BigQuery

Which repos are affected by our vulnerability disclosure (Operation Rosehub)?

```
SELECT pop, repo_name, path
FROM (
  SELECT id, repo_name, path
  FROM `bigquery-public-data.github_repos.files` AS files
  WHERE path LIKE '%pom.xml' AND
    EXISTS (
      SELECT 1
      FROM `bigquery-public-data.github_repos.contents`
      WHERE NOT binary AND
        content LIKE '%commons-collections<%' AND
        content LIKE '%>3.2.1<%' AND
        id = files.id
    )
)
JOIN (
  SELECT
    difference.new_sha1 AS id,
    ARRAY_LENGTH(repo_name) AS pop
  FROM `bigquery-public-data.github_repos.commits`
  CROSS JOIN UNNEST(difference) AS difference
)
USING (id)
ORDER BY pop DESC;
```

Checkout the work of Felipe Hoffa

GHTorrent

GHTorrent collects all data, both events and entities, from the GitHub REST API and makes them available:

- As queriable MySQL and MongoDB databases
- As database dumps for both databases
- Over Google BigQuery (MySQL data only)
- As continuously updating data streams (also, over Google Pub/Sub)

Some statistics about GHTorrent

- > 15TB of MongoDB (raw) data
- > 5B rows in MySQL
- 140k API reqs/hour
- 70+ users donated API keys
- 300 users, from > 200 institutions have access
- 100+ papers
- > 40% of all GitHub-related papers (Cosentino et al.)

Growth

This is how fast MongoDB grows (× 1M)

Entity	2013	Feb 2016	Apr 2017	Δ 2016 - 2017
Events	43	476	886	1.9×
Users	0.7	6.7	13.5	2×
Repos	1.3	28	57	2×
Commits	29.9	367	662	1.8×
Issues	2.3	24.1	41.1	1.7×
Pull requests	1.1	11.9	23.8	2×
Issue Comments	2.8	42	74.2	1.7×
Watchers	7.7	51	84.9	1.6×

GitHub/GHTorrent doubled its size during the last 14 months!

Data collection

GHTorrent follows the event stream and recursively retrieves linked entities.

Distributed operation

- *Event retrieval* nodes query the event API
- *Data retrieval* nodes apply the recursive descent retrieval process
  - One API key per data retrieval node

Collection modes

- **Normal operation**: Follow the event timeline, apply dependency-based retrieval

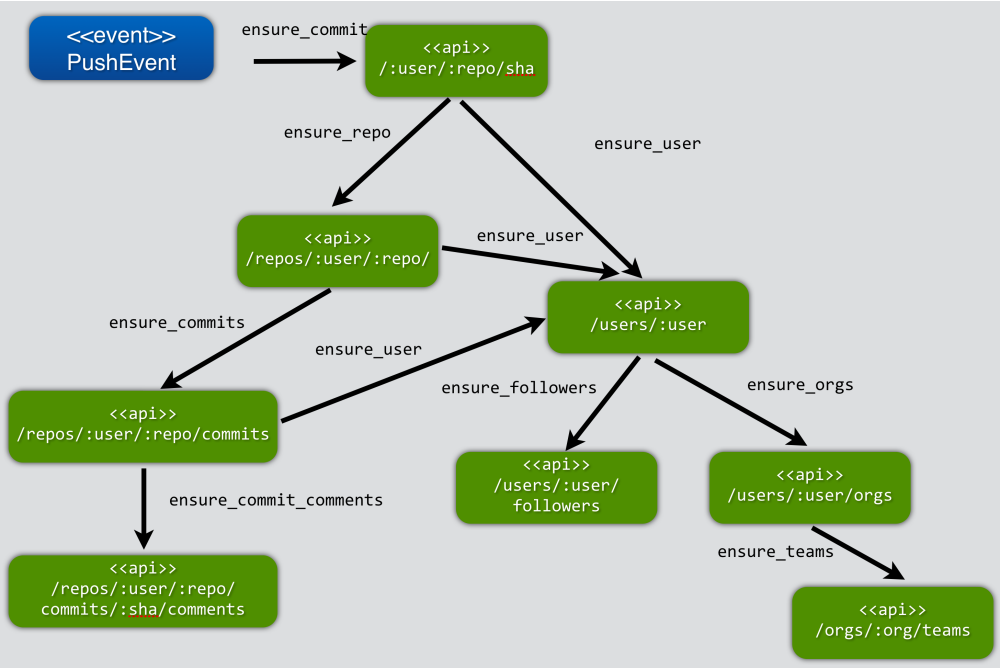

Figure 2: Retrieval scheme

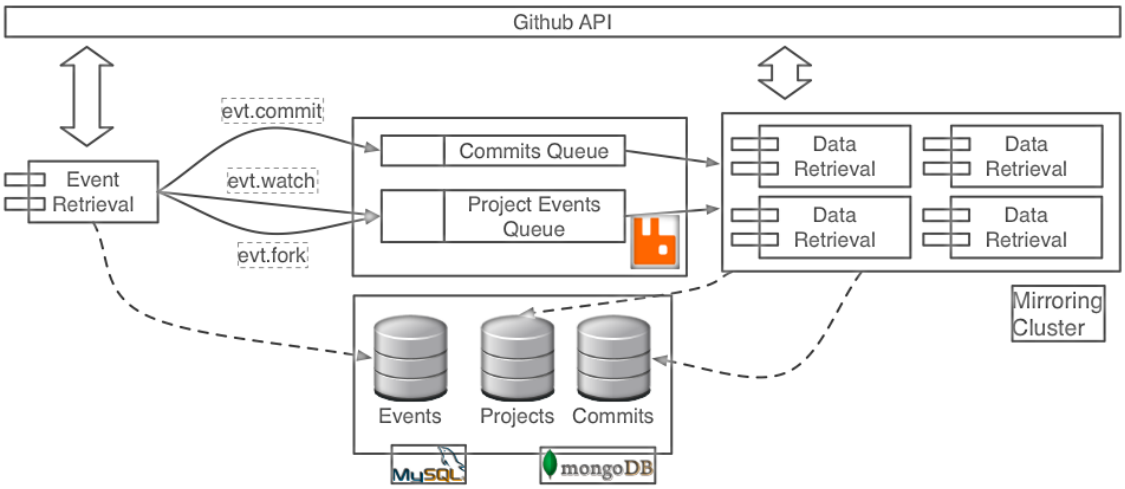

Figure 3: Retrieval architecture

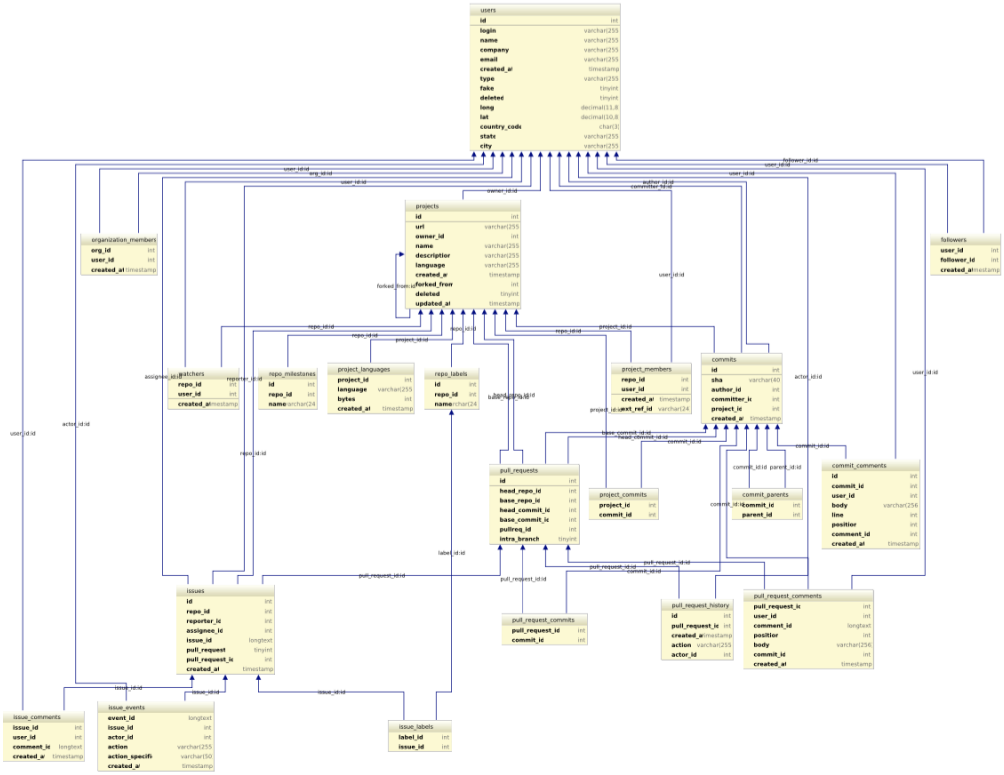

Figure 4: The MySQL schema

- **Periodic updates**: Refresh the state of all repos/users (cater for deleted repos, changed user locations etc)
- **Full retrievals**: Get all info for a repo/user, in case some information is missing

The MySQL schema

Important tables – Users

- Type: USR or ORG
- Fake users represent emails with corresponding GitHub account
- ~15% users are geo-located
- Emails and real names not distributed by default

Important tables – Projects & Commits

- A fork has a non-NULL `forked_from` and `forked_commit_id`
- `language` in projects represents the primary language at collection time. More details in table `project_languages`
- A commit is only stored once per SHA
- The table `project_commits` links commits to projects

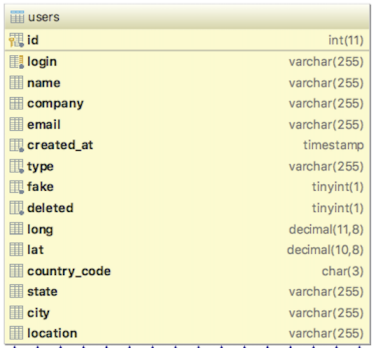

Figure 5: Users table

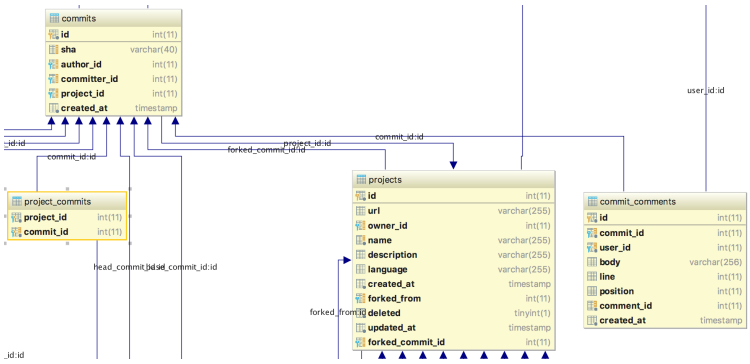

Figure 6: Projects and commits tables

Important tables – Pull Requests

- If `head_repo_id` is null → the fork was deleted
- Entries in `pull_request_history` represent a log of actions on a PR
- PRs and issues are dual on GitHub
  - The PR history is also reflected in `issue_events`
- A PR is associated with 3 times of comments:
  - `issue_comments`: The PR/Issue discussion
  - `pull_request_comments`: PR-global code review comments
  - `commit_comments`: Line-local code review comments
- A PR has associated `pull_request_commits` (not retrievable in case of rebases)

SQL Schema: Things to remember

- The info in `organization_members` and `project_members` is not trustworthy, as it is optional for orgs/projects to publish it
- `watchers` means GitHub stars
- `watchers` and `followers` are append-only tables (GitHub does not emit events on delete/update)
- `followers` is only updated during periodic updates, as GitHub stopped emitting `FollowEvents`
- The timestamps in `followers` and `watchers` were relative to the time of retrieval. Since mid-2015 they are accurate.

Querying the SQL schema

Let's query!

To make queries shorter, we use the following project

```
select p.id, u.login, p.name
from projects p, users u
where u.id = p.owner_id
  and u.login = 'ReactiveX'
  and p.name = 'rxjs'
```

Table 2: 1 records

id	login	name
1	ReactiveX	rxjs

Get all commits

```
select u.login as author, c.sha, c.created_at
from users u, commits c, project_commits pc
where pc.commit_id = c.id
  and pc.project_id = 1
  and u.id = c.author_id
order by c.created_at desc
```

Table 3: Displaying records 1 - 5

author	sha	created_at
blesh	996bf6d8ae8bedc6ca4ed9daf2c3d6e0244bfbcd	2017-02-13 05:21:58.000000
trxcllnt	22e4c17046331cd1d673a24305f79c90278e96b3	2017-02-13 05:20:10.000000
trxcllnt	c02bc8fb4766296abf4cfcde5d63e7451b5d2581	2017-02-12 06:53:06.000000
martinsik	db125e6c8673bc3417029b7a616c43d775726fdd	2017-02-11 13:02:09.000000
jayphelps	31dfc7324bf51b98df6678f072b7a3bd4c0b5888	2017-02-09 18:57:47.000000

Basic info about forks

```
select u.login, p.name,  p.forked_commit_id
from projects p, users u
where p.forked_from is not null
and u.id = p.owner_id
order by p.id desc
```

Table 4: Displaying records 1 - 5

login	name	forked_commit_id
Shahor	rxjs	1
sergonius	rxjs	1
luwenxull	rxjs	1
joseph-ortiz	rxjs	3

login	name	forked_commit_id
vyorkin-forks	rxjs	3

Get all commits for a fork (1)

```
select c.id, u1.login as author, c.sha, c.created_at
from projects p, users u, users u1, commits c, project_commits pc
where pc.commit_id = c.id
  and pc.project_id = p.id
  and u1.id = c.author_id
  and u.id = p.owner_id
  and u.login = 'herflis'
  and p.name = 'rxjs-1'
order by c.created_at desc
```

Table 5: 2 records

id	author	sha	created_at
4002	herflis	46c954a431294fb87cbd791368d0d701a9446eba	2017-02-10 15:56:51.000000
3	jayphelps	31dfc7324bf51b98df6678f072b7a3bd4c0b5888	2017-02-09 18:57:47.000000

Q: Why do we only get 2 commits?

Get all commits for a fork (2)

What the previous query returns is a list of commits up to the fork point. We also need to retrieve the base repository commits up to the fork point.

First, we get some information to make querying simpler

```
select p.id, p.forked_from, p.forked_commit_id
from projects p, users u
where u.id = p.owner_id
  and u.login = 'herflis'
  and p.name = 'rxjs-1'
```

Table 6: 1 records

id	forked_from	forked_commit_id
450	1	3

Get all commits for a fork (3)

```
select id, author, sha, created_at
from (
  select c.id, u1.login as author, c.sha, c.created_at
  from users u1, commits c, project_commits pc
  where pc.commit_id = c.id
```

```
    and pc.project_id = 450
    and u1.id = c.author_id
  union
  select c.id, u1.login as author, c.sha, c.created_at
  from users u1, commits c, project_commits pc
  where pc.commit_id = c.id
    and pc.project_id = 1
    and u1.id = c.author_id
    and c.created_at <
      (select created_at from commits where id = 3)
) as commits
order by created_at desc
```

Table 7: Displaying records 1 - 5

id	author	sha	created_at
4002	herflis	46c954a431294fb87cbd791368d0d701a9446eba	2017-02-10 15:56:51.000000
3	jayphelps	31dfc7324bf51b98df6678f072b7a3bd4c0b5888	2017-02-09 18:57:47.000000
5	kwonoj	01a2d15ef8e1795d04a5b535acb97068feb56385	2017-02-09 05:36:55.000000
4	kwonoj	6ce4773e3894b3718dfc61edd6c38510b9a1e77b	2017-02-05 20:23:45.000000
7	Podlas29	7eb50152b00f3c226cad0f10203d22a813e50b2b	2017-02-05 13:29:44.000000

Which forks contributed code?

```
select cr.month as month, cr.created, co.contributing
from (
  select date(p.created_at,'start of month','+1 month','-1 day')  as month,
         count(*) as created
  from projects p
  where p.forked_from = 1
  group by month) as cr
join
  (select date(p.created_at,'start of month','+1 month','-1 day') as month,
         count(*) as contributing
  from projects p
  where p.forked_from = 1
  and exists (
    select *
    from pull_requests pr
    where pr.head_repo_id = p.id)
  group by month) as co
on cr.month = co.month
order by month desc
```

Table 8: Displaying records 1 - 5

month	created	contributing
2017-02-28	16	3
2017-01-31	42	10
2016-12-31	49	12
2016-11-30	24	5

month	created	contributing
2016-10-31	27	7

What are the core team members?

```
select distinct(u.login) as login
    from commits c, users u, project_commits pc
    where u.id = c.committer_id
      and u.fake is 0
      and pc.commit_id = c.id
      and pc.project_id = 1
      and c.created_at > datetime(date('2017-02-14'), '-3 month')
union
select distinct(u.login) as login
  from pull_requests pr, users u, pull_request_history prh
  where u.id = prh.actor_id
    and prh.action = 'merged'
    and prh.pull_request_id = pr.id
    and pr.base_repo_id = 1
    and prh.created_at > datetime(date('2017-02-14'), '-3 month')
```

Table 9: Displaying records 1 - 5

login
Brooooooklyn
Dorus
Podlas29
blesh
feloy

Which countries contributed most commits?

```
select u.country_code, count(*) num_commits
from users u, commits c
where u.id = c.author_id
  and country_code is not null
group by u.country_code
order by num_commits desc
```

Table 10: Displaying records 1 - 5

country_code	num_commits
us	1070
fi	820
jp	195
br	155
fr	53

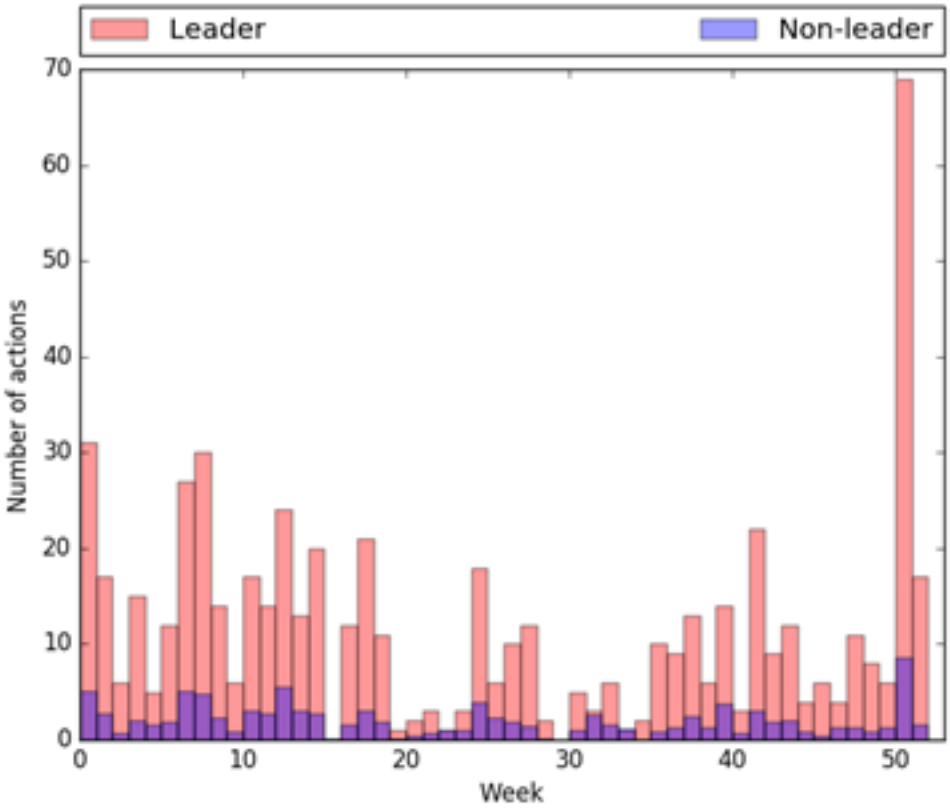

Figure 7: Leader vs member contributions

Tips for using GHTorrent effectively

- Use MySQL for queries
  - Downloading and install a local copy is **recommended**
- Only use MongoDB when you need:
  - Text for issue comments, commit messages, commit diffs
  - Connect to the online service
  - Don't do aggregations on MongoDB, prefer index-based queries
- Use BigQuery for large aggregations
  - Also possible to combine with the GitHub dataset

Real-world study example

- Complex
- Expensive
- Example: leader contribution

Project selection

- Examine data for a period of exactly one year
- Eliminate projects that started their life within the examined period

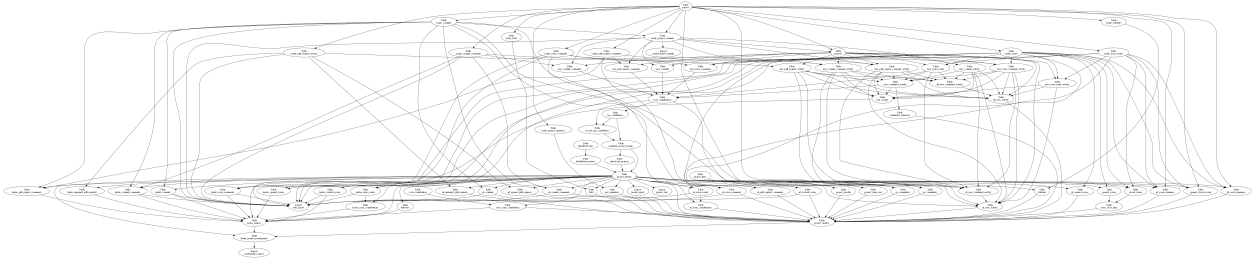

Figure 8: SQL query dependency graph

- Select projects having at least ten watchers and at least 20 commits made by at least seven different people

Project leaders

- Member's contribution:
- Commits
- Commit comments
- Issue events
- Issue comments
- Pull request comments
- Leader: the individual with 20% more project contributions than the one with second highest

Real-world queries can be complex

(2600 lines of SQL)

Real-world queries can be expensive

```
May 14 14:37 yearly_commits
May 14 16:34 yearly_project_commits
May 15 00:02 projects
May 15 00:04 user_commit_activity
May 15 01:57 yearly_commit_comments
[22 lines omitted]
May 15 03:28 commits_with_comments
May 15 04:50 contributor_followers
[42 lines omitted]
May 15 08:49 q1_committers
May 15 10:02 q1_issue_managers
[7 lines omitted]
May 15 10:51 nl_issues_leader_comments
```

What can we do?

- Modularize
- Incremental construction
- Unit testing
- Execution checkpoints

- Reuse

Approaches

- Oracle/DB2/PostgreSQL/... materialized views
- Justin Swanhart's MySQL Flexviews
- Make-based simple-rolap

Example task

1. Choose repositories that have forks (A)
2. from A, exclude repos that never received a PR (B)
3. from B, exclude repos that were inactive *recently* (C)
4. from C, exclude repos that have fewer than 100 stars (D)
5. Obtain number of files and lines of code for each project

We could then apply further criteria (e.g. programming language, build system, number of contributors, etc).

Environment setup

- Clone and install github.com/dspinellis/rdbunit
- Clone github.com/dspinellis/simple-rolap

Project setup

Create a `Makefile` with the following contents:

```
export RDBMS?=sqlite
export MAINDB?=rxjs-ghtorrent
export ROLAPDB?=stratsel
export DEPENDENCIES=rxjs-ghtorrent.db

include ../../Makefile

rxjs-ghtorrent.db:
        wget https://github.com/ghtorrent/tutorial/raw/master/rxjs-ghtorrent.db
```

Repositories with forks

Create a file `forked_projects.sql`

```
-- Projects that have been forked

create table stratsel.forked_projects AS
  select distinct forked_from as id from projects;
```

Run it

```
$ make
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
mkdir -p tables
sh ../..//run_sql.sh forked_projects.sql >tables/forked_projects
```

Run it again

```
$ make
make: Nothing to be done for 'all'.
```

Yes, but is it correct?

Create a file `forked_projects.rdbu`

```
BEGIN SETUP

projects:
id      forked_from
1       15
2       15
3       10
4       NULL

END

INCLUDE CREATE forked_projects.sql

BEGIN RESULT
stratsel.forked_projects:
id
15
10
END
```

Run the tests

```
$ make test
../..//run_test.sh
not ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
```

Houston, we have a problem!

Step-by–step debugging

```
$ rdbunit --database=sqlite forked_projects.rdbu >script.sql
$ sqlite3
SQLite version 3.8.7.1 2014-10-29 13:59:56
sqlite> .read script.sql
```

```
not ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1

sqlite> select * from test_stratsel.forked_projects;
15
10

sqlite> select count(*) from test_stratsel.forked_projects;
3
```

Correct the error

```
-- Projects that have been forked

create table stratsel.forked_projects AS
  select distinct forked_from as id from projects
  where forked_from is not null;
```

Test again

```
$ make test
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
```

Bingo!

Exclude projects with no PRs

Create a file `pr_projects.sql`

```
-- Projects that have been forked and have a PR

create table stratsel.pr_projects AS
  select distinct forked_projects.id from stratsel.forked_projects
  inner join issues on issues.repo_id = forked_projects.id;
```

Run it (incrementally)

```
$ make
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
mkdir -p tables
sh ../..//run_sql.sh pr_projects.sql >tables/pr_projects
```

Test a little

Create a file `pr_projects.rdbu`

```
# Projects that have at least one PR associated with them

BEGIN SETUP

stratsel.forked_projects:
id
1
2
3
4

issues:
id      repo_id
15      1
16      1
17      4
END

INCLUDE CREATE pr_projects.sql

BEGIN RESULT
stratsel.pr_projects:
id
1
4
END
```

Run tests

```
$ make test
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
```

Exclude projects with no recent commits

Create a file `recent_commit_projects.sql`

```sql
-- Projects with recent commits

create table stratsel.pr_projects AS
  select distinct pr_projects.id
  from stratsel.pr_projects
  left join commits
  on commits.project_id = pr_projects.id
  where created_at > '2017-01-01';
```

Test a little (really now?)

Create a file `recent_commit_projects.rdbu`

```
# Projects that have a recent commit associated with them

BEGIN SETUP
stratsel.pr_projects:
id
1
2
3
4

commits:
id      project_id      created_at
15      1               '2017-05-12'
16      1               '2010-01-01'
16      2               '2017-01-02'
16      2               '2017-01-03'
17      4               '1970-01-01'
END

INCLUDE CREATE recent_commit_projects.sql

BEGIN RESULT
stratsel.recent_commit_projects:
id
1
2
END
```

Run tests

```
$ make test
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
Error: near line 25: table pr_projects already exists
Error: near line 38: no such table: test_stratsel.recent_commit_projects
1..1
../../Makefile:79: recipe for target 'test' failed
make: *** [test] Error 1
```

???

Correct table name in query

```
-- Projects with recent commits

create table stratsel.recent_commit_projects AS
```

```
  select distinct pr_projects.id
  from stratsel.pr_projects
  left join commits
  on commits.project_id = pr_projects.id
  where created_at > '2017-01-01';
```

Run tests again

```
$ make test
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
```

Incremental build

```
$ make
mkdir -p tables
sh ../..//run_sql.sh recent_commit_projects.sql >tables/recent_commit_projects
```

An aha moment

- PR means "pull request" not "problem report"
- Rewrite `pr_projects.sql`

```
-- Projects that have been forked and have a PR

create table stratsel.pr_projects AS
  select distinct forked_projects.id from stratsel.forked_projects
  inner join pull_requests on pull_requests.base_repo_id = forked_projects.id;
```

Run make again

Notice that only dependent tables get built

```
$ make
mkdir -p tables
sh ../..//run_sql.sh pr_projects.sql >tables/pr_projects
mkdir -p tables
sh ../..//run_sql.sh recent_commit_projects.sql >tables/recent_commit_projects
```

Exclude projects with no recent issues

Test and run

```
$ make test
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
ok 1 - recent_issue_projects.rdbu: test_stratsel.recent_issue_projects
1..1
$ make
mkdir -p tables
sh ../..//run_sql.sh recent_issue_projects.sql >tables/recent_issue_projects
```

Count number of project stars

Create a file `project_stars.sql`

```
-- Projects with recent issues and the number of stars

create table stratsel.project_stars AS
  select recent_issue_projects.id as id, count(*) as stars
  from stratsel.recent_issue_projects
  left join watchers
  on watchers.repo_id = recent_issue_projects.id
  group by recent_issue_projects.id;
```

Corresponding test

Create a file `project_stars.rdbu`

```
# Projects with recent issues and the number of stars

BEGIN SETUP
stratsel.recent_issue_projects:
id
1
2
3

watchers:
repo_id
1
1
2
END

INCLUDE CREATE project_stars.sql

BEGIN RESULT
stratsel.project_stars:
```

```
id      stars
1       2
2       1
3       0
END
```

Run test

```
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
not ok 1 - project_stars.rdbu: test_stratsel.project_stars
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
ok 1 - recent_issue_projects.rdbu: test_stratsel.recent_issue_projects
1..1
```

Fix query

Count only non-null rows

```
-- Projects with recent issues and the number of stars

create table stratsel.project_stars AS
  select recent_issue_projects.id as id, count(watchers.repo_id) as stars
  from stratsel.recent_issue_projects
  left join watchers
  on watchers.repo_id = recent_issue_projects.id
  group by recent_issue_projects.id;
```

Run test again

```
$ make test
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
ok 1 - project_stars.rdbu: test_stratsel.project_stars
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
ok 1 - recent_issue_projects.rdbu: test_stratsel.recent_issue_projects
1..1
```

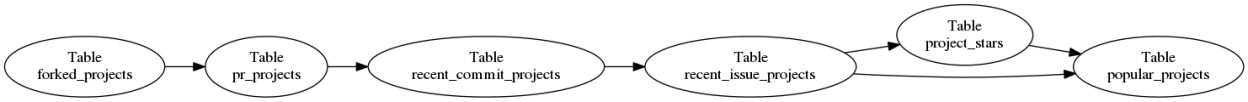

Figure 9: SQL query simple dependency graph

Add and test popular projects

```
$ make test
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - popular_projects.rdbu: test_stratsel.popular_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
ok 1 - project_stars.rdbu: test_stratsel.project_stars
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
ok 1 - recent_issue_projects.rdbu: test_stratsel.recent_issue_projects
1..1
$ make
mkdir -p tables
sh ../..//run_sql.sh project_stars.sql >tables/project_stars
mkdir -p tables
sh ../..//run_sql.sh popular_projects.sql >tables/popular_projects
```

Where do we stand?

```
$ make graph.png
../..//dep2dot.sed .depend >graph.dot
dot -Tpng graph.dot -o graph.png
```

Obtain repo data

- Write out repository URLs
- Create script to analyze repos
- Import back results for further analysis

Repository URLs

Create a file `project_urls.sql`

```
-- URLs of popular projects
select projects.id, 'https://github.com/' || substr(url, 30) as url
from stratsel.popular_projects
left join projects
on projects.id = popular_projects.id;
```

Unit test

- Use `INCLUDE SELECT`
- No table name in result

```
# Projects that have a recent commit associated with them

BEGIN SETUP
stratsel.popular_projects:
id
1
2

projects:
id      url
1       'https://api.github.com/repos/foo'
2       'https://api.github.com/repos/bar'
3       'https://api.github.com/repos/foobar'
END

INCLUDE SELECT project_urls.sql

BEGIN RESULT
id      url
1       'https://github.com/foo'
2       'https://github.com/bar'
END
```

Run tests

```
$ make test
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
../..//run_test.sh
ok 1 - forked_projects.rdbu: test_stratsel.forked_projects
1..1
ok 1 - popular_projects.rdbu: test_stratsel.popular_projects
1..1
ok 1 - pr_projects.rdbu: test_stratsel.pr_projects
1..1
ok 1 - project_stars.rdbu: test_stratsel.project_stars
1..1
ok 1 - project_urls.rdbu: test_select_result
1..1
ok 1 - recent_commit_projects.rdbu: test_stratsel.recent_commit_projects
1..1
ok 1 - recent_issue_projects.rdbu: test_stratsel.recent_issue_projects
1..1
```

Run queries

- Result is in `reports`
- Named after the query

```
$ make
rm -f ./.depend
sh ../..//mkdep.sh >./.depend
mkdir -p reports
sh ../..//run_sql.sh project_urls.sql >reports/project_urls.txt
$ cat reports/project_urls.txt
1|https://github.com/ReactiveX/rxjs
```

Script to analyze repos

```
#!/bin/sh
# Create a CSV file with files and lines per project

set -e

mkdir -p clones data
cd clones

file_list()
{
  git ls-tree --full-tree -r --name-only "$@" HEAD
}

while IFS=\| read id url ; do
  git clone --bare "$url" $id
  cd $id
  nfiles=$(file_list | wc -l)
  nlines=$(file_list -z |
    xargs -0 -I '{}' git show 'HEAD:{}' |
    wc -l)
  echo "$id,$nfiles,$nlines"
  cd ..
done <../reports/project_urls.txt >../data/metrics.csv
```

Import data (SQLite)

```
Create file `project_metrics.sql`
create table stratsel.project_metrics (id integer,
  files integer, lines integer);
.separator ","
.import data/metrics.csv stratsel.project_metrics
```

Import data (MySQL)

```
Create file `project_metrics.sql`
create table stratsel.project_metrics (id integer,
  files integer, lines integer);
LOAD DATA LOCAL INFILE 'data/metrics.csv'
```

```
INTO TABLE stratsel.project_metrics
FIELDS TERMINATED BY ',';
```

Add dependencies

```
tables/project_metrics: data/metrics.csv

data/metrics.csv: reports/project_urls.txt project_metrics.sh
        sh project_metrics.sh
```

Run make

```
$ make
sh project_metrics.sh
Cloning into bare repository '1'...
remote: Counting objects: 30680, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 30680 (delta 2), reused 2 (delta 2), pack-reused 30671
Receiving objects: 100% (30680/30680), 73.95 MiB | 10.29 MiB/s, done.
Resolving deltas: 100% (23964/23964), done.
Checking connectivity... done.
mkdir -p tables
sh ../..//run_sql.sh project_metrics.sql >tables/project_metrics
```

Other goodies

- `make tags`
- `make sorted-dependencies`
- `make clean`

Takeaways of example study

- Split analysis into many small queries
- Use simple-rolap to automate process
- *Errare humanum est*
- Code a little, test a little
- Use rdbunit to express tests
- Write additional processing as scripts
- Use *make* dependencies to tie everything together

Combining MongoDB and MySQL data

"Use the `bash`, Luke"

```
echo "select c.sha \
      from commits c, project_commits pc \
      where pc.commit_id = c.id and \
      pc.project_id = 1;" |
sqlite3 rxjs-ghtorrent.db|
```

```
while read sha; do
  echo "db.commits.find({sha: '$sha'},{'commit.message':1, _id:0})" |
  mongo --quiet github|
  jq '.commit.message'|
  xargs echo $sha
done
```

- MySQL is basically a 2D index to MongoDB
- Most useful information in MongoDB is in the `commits` collection

Use BigQuery to join public datasets

```
SELECT c.country, COUNT(*) pushes,
  FIRST(c.population) population,
  1000*COUNT(*)/FIRST(c.population) ratio_pushes
FROM [githubarchive:month.201608] a
JOIN [ghtorrent-bq:ght_2017_05_01.users] b
ON a.actor.login=b.login
JOIN (
  SELECT LOWER(iso) iso, population, country
  FROM [gdelt-bq:extra.countryinfo]
  WHERE population > 300000
) c
ON c.iso=b.country_code
WHERE country_code != '\\N'
AND a.type='PushEvent'
GROUP BY 1
ORDER BY ratio_pushes DESC
```

Real-time data analysis

```
conn = Bunny.new(:host => '127.0.0.1', :port => 5672,
                 :username => 'streamer', :password => 'streamer')
conn.start
ch = conn.create_channel
exchange = ch.topic('ght-streams', :durable => true)
q = ch.queue('gousiosg_queue', :auto_delete => true)
q.bind(exchange, :routing_key => 'ent.commits.insert')

q.subscribe do |delivery_info, properties, payload|
  commit = JSON.parse(payload)
  repo = commit['url'].split(/\//)[4..5].join('/')
  commit['files'].each do |f|
    next unless /passw[or]*d.*[:=].*/.match? f['patch']
    puts "Password found! repo=#{repo}, sha=#{commit['sha']}"
  end
end
```

Get updates from GHTorrent in a streaming fashion.

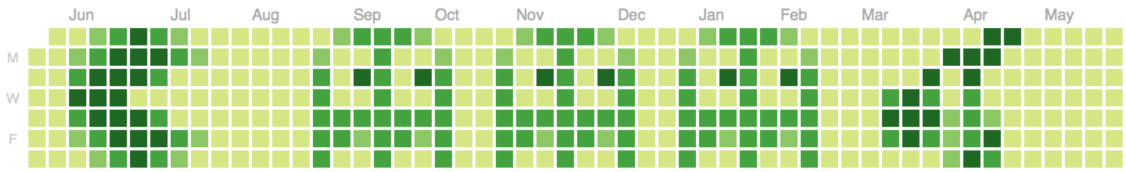

Figure 10: Retrieval scheme

Avoiding common pitfalls

- GHTorrent (or any GitHub dataset) is not an exact GitHub replica

You can retrieve a set of data and write scripts to fill in the gaps using the GitHub API/GraphQL.

- Don't select repositories to analyze manually

This will lead to non-representativeness. Define a population first, then use *stratified sampling* to select repositories from it.

Avoiding common pitfalls

- Repositories are not projects

A large project, such as Homebrew, is usually composed of multiple repositories. Some repositories have equally active forks. To study a *project* as a community, make sure you include all its related activity.

- Most repositories are idle

Selecting randomly from repositories will lead to toy/dead repositories.

Avoiding common pitfals

- Many active projects do not use GitHub exclusively

The prime example are Apache projects. Projects with with high commit counts but not so many issues/pull requests are using other collaboration platforms.

- Not all activity is due to users

Beware of bots and tool integrations, especially in issues and pull requests!

Avoiding common pitfals

- At least 15% of merged pull requests appear as not merged

Use the following heuristics to uncover them

1. An entry in table `pull_request_commits` also appears in `project_commits`
2. A commit in `project_commits` references a PR number (e.g. `fixes: 12`)
3. One of the 3 last comments in the discussion match following regex `(?:merg|appl|pull|push|integrat)(?:ing|i?ed)`
4. The above regex matches the last comment before closing the PR

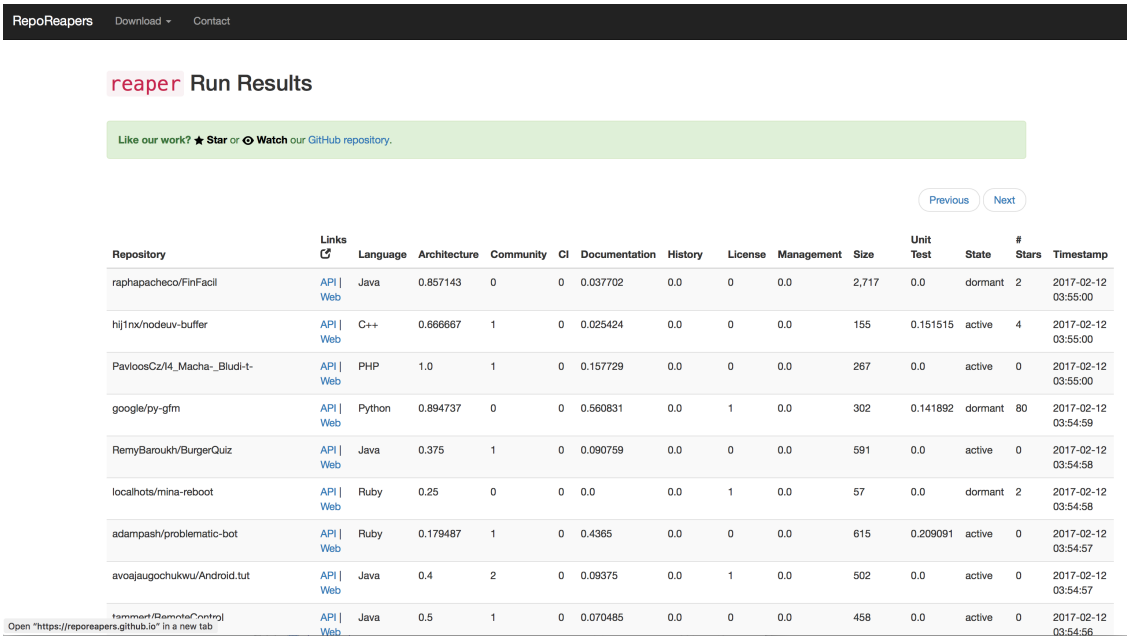

Figure 11: Reporeapers

Selecting repositories to analyze

Generally, we need repositories that are active and representative. We can use the following selection process:

From the whole population:

1. Choose repositories that have forks (A)
2. from A, exclude repos that where inactive *recently* (B)
3. from B, exclude repos that never received a PR (C)

On C, we can then apply further criteria (e.g. programming language, build system, minimum number of stars etc).

If the project list is still too long, then apply *stratified sampling*

Modular querying and testing

- Modular queries with simple-rolap
- Unit testing with rdbunit

Reporeapers

A curated set of repositories worth analyzing

Legal issues


```
{
    "id": "4141500869",
    "type": "IssueCommentEvent",
```

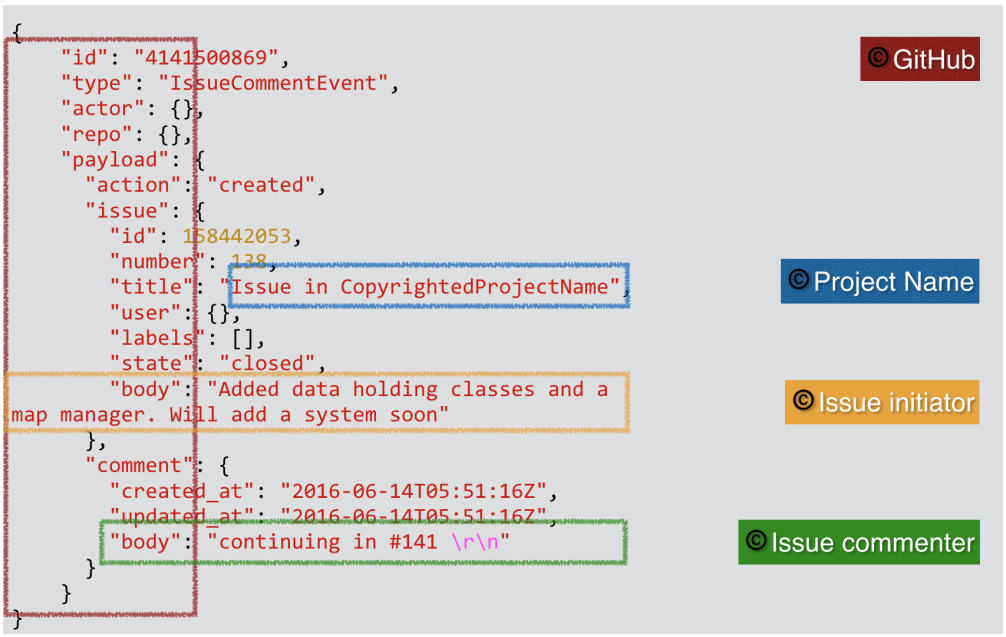

Figure 12: Copyright


```
    "actor": {},
    "repo": {},
    "payload": {
      "action": "created",
      "issue": {
        "title": "Issue in CopyrightedProjectName",
        "user": {},
        "labels": [],
        "body": "Added data holding classes and a map manager. Will add a system soon"
      },
      "comment": {
        "created_at": "2016-06-14T05:51:16Z",
        "body": "continuing in #141 \r\n"
      }
    }
}
```


Q: How many copyrights do you see in the JSON above?

Copyright situation

- We can only assert copyright for non-trivial works
- GHTorrent has a CC-BY-SA license

Privacy

> **Privacy** is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively.

To comply with the law, GHTorrent does not distribute privacy-sensitive information by default.

- Still available by filling in the Personal data form
- Please don't spam developers
- Several ethical considerations, perhaps review your research by your ethics board

How to run GHTorrent locally?

Using plain Ruby

```
gem install sqlite3 bundler
git clone https://github.com/gousiosg/github-mirror
cd github-mirror
bundle install
mv config.yaml.standalone > config.yaml
ruby -Ilib bin/ght-retrieve-repo -t token rails rails
```

Using Vagrant

```
apt-get install vagrant
git clone https://github.com/ghtorrent/ghtorrent-vagrant.git
cd ghtorrent-vagrant
vagrant ssh
ruby -Ilib bin/ght-retrieve-repo -t token rails rails
```

Contributing to GHTorrent

We need your help!

- Do you have spare hacking time? Pick a task below:
  - Retrieve all commits for non-forked repos
  - Pull request labels
  - Reactions, Issue Timelines
  - More info for repos: licenses, language updates etc
- MongoDB data on BigQuery / Spark
- Do you have a derivative service/dataset? Let's link it!
- Did you find a bug? Let's discuss how to fix it!
- Did you write a paper? Get it included in the Hall of fame!

Don't hesitate to contact us: @gousiosg, @coolsweng

Further reading

[1] G. Gousios, "The GHTorrent dataset and tool suite," in *Proceedings of the 10th working conference on mining software repositories*, 2013, pp. 233–236.

[2] G. Gousios and D. Spinellis, "GHTorrent: GitHub's data from a firehose," in *MSR '12: Proceedings of the 9th working conference on mining software repositories*, 2012, pp. 12–21.

[3] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[4] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, pp. 1–35, 2017.

Copyright