Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

ARON METZIG

BACHELOR THESIS

IMPLEMENTATION OF A GITLAB ADAPTER AND THE EVOLUTION OF IT'S INTERFACE

Submitted on 4 February 2019

Supervisor: Maximilian Capraro M.Sc.; Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 4 February 2019

License

Erlangen, 4 February 2019

Card of thanks

For always kind and supportive suggestions, not only in code contributions.
A special thanks for Maximilian Capraro, who made this thesis possible.

Abstract

Gitlab is an Software forge with integrated Continuous Integration (CI) and DevOps features under the MIT license for the non commercial version. As Gitlab is broadly used for inner source software development it is important to be able to measure code collaboration within existing ecosystems. To measure the code collaboration, the Patch-Flow Crawler (FPC) was created by the Open Source Research Group's (OSR). It is not yet possible to use this software with Gitlab. Therefore this thesis implements an adapter that fetches the relevant patches from the Gitlab-API and transforms it to the FPC can calculate the patch-flow. This includes organizational structures, mapping of pseudonyms to identities and the ownership of patches. The implementation was evaluated with the use of the production RRZE-Gitlab instance. Also Unit and Integration-tests validate the correctness of the code after Gitlab-API changes. The FPC now can natively calculate the patch-flow from an configured Gitlab instance and is able to to determine entity-relationships, if the Gitlab instance contains the reliable data.

Contents

1 Introduction **1**

2 Requirements **3**
2.1 Functional Requirements 3
2.2 Non functional Requirements 4
2.3 Evaluation 4

3 Implementation **6**
3.1 Design of the SDK 6
3.1.1 GitlabService – Authorization and endpoints 6
3.1.2 GitlabClient - Fetching Data from Gitlab 7
3.1.3 GitlabPaginator – Utilization of the Gitlab-API 8
3.1.4 GitlabCommitIterator – Effectively fetching Patches 9
3.1.5 Transformer – Enter the PFC world 10
3.1.6 GitlabAdapter- Setting it all together 11
3.2 Plugin Implementation 13
3.2.1 Configuration 13
3.2.2 AddRepositoryPreStep – Setting up repositories 14
3.2.3 GitlabAdapter - Fetching the Patches 14
3.2.4 ResolvePseudonymProcessor – Reuse pseudonyms 14
3.2.5 GitlabAddAuthorProcessor – Map new Users 15
3.3 Manual mapping pseudonyms 15
3.3.1 Webclient - Provide functionality to the user 15
3.3.2 The backend logic 17
3.3.3 Getting unmapped pseudonyms 17
3.3.4 Manually mapping a pseudonym 18
3.3.5 Creating the organizational structure 18
3.3.6 Logical inference for mapping Authors to Patches 19
3.4 Testing 19
3.4.1 Setting up an testing environment 19
3.4.2 Creation of an Docker backup 20
3.4.3 Inserting testing data 21

3.4.4 Integration in the Pipeline 22

4 Evaluation **23**
4.1 Performance and testing system 23
4.2 Database and validity 23
4.3 Logging and resilience 25
4.4 Testing 25

Appendices **27**
Appendix A UML Gitlab 27
Appendix B Transformer code 28
Appendix C Excerpt Person 29
Appendix D Excerpt InnerSourceProject 30
Appendix E Excerpt Filechange 31
Appendix F Excerpt Patch 32
Appendix G Excerpt Repository 33
Appendix H API-Key retrievemnt 34
Appendix I Warn Log 35
Appendix J Error Log 37

References **40**

1 Introduction

Inner source code collaboration is the use of open source practices and the establishment of open source-like communities for firm internal development (Capraro and Riehle, 2017). This Inner source collaboration is measurable due the patch-flow-method.
In OS, a patch is a code contribution from an individual to an OS project.
The patch-flow then is the flow of such patches in across organizational boundaries. Hence the PFC, that measures this code collaboration between different inner source teams and projects using this flow. The collaboration is measured by creating a directed weighted graph between the organizational units. (Capraro, Dorner and Riehle, 2018)

For hosting inner source projects industry uses software forges like Github Enterprise, Gitlab, Bitbucket and home made solutions.
For beeing able to migrate the PFC into a company, it is necessary that it is able to interact with the existing software forge.
Gitlab is a broadly used software forge, which includes industry-standards features, like kubernets and native Continuous Integration (CI) support . The acquisition of Microsoft emphasizes it's distribution.

So the ability of supporting Gitlab it is an important milestone for the FPC.
This thesis is implementing an adapter between the Gitlab-API and the PFC. The PFC is the demanding part, where the adapter itself is implemented as an plugin.
With this implementation the FPC is now able to extract the necessary data from Gitlab, transforming the data into the internal entities and finally measures the given patch-flow.
The following section will describe the given requirements, including an in-depth analysis of the recent Gitlab-API and the FPC.
Then the implementation section will cover the creation of an Gitlab test infrastructure and the implementation of the adapter itself.

The evaluation of this extension is using the real world Gitlab instance of the RRZE, which is the service provider of the Friedrich-Alexander University. The contributions of this thesis are:

- An adapter that allows to crawl the patch-flow from the Gitlab-API
  - Authentification for the Gitlab-API
  - Pagination handling of the Gitlab-API
  - Fetching for new projects
  - Fetching all patches in a given time range
  - Fetching all file-changes of a given commit
  - Fetching and mapping persons from Gitlab
  - Assigning a person to a patch, when this is logical inferable
  - Bugfixing for certain Gitlab versions
- Manual mapping of patch pseudonyms to persons
  - Creating frontend components
  - Pseudonyms querying logic
  - Communication due REST
- A test setup for this adapter
- Setting up a local Gitlab instance for a CI pipeline
- Dockerfile that automatically sets up the Gitlab-instance
  - Gitlab configuration
  - API-Key
  - Mock data
- Integration test, for all critical API-Calls
- Unit tests

The thesis is structured as follows. Section one covers the specifications for the adapter.
Section two on the implementation itself and section three evaluates the fulfillment of the specifications.

2 Requirements

The PFC is a sophisticated toolbox, which is able to interact with multiple software forges and provides an web frontend which communicates over a REST-service with the backend. To handle this complexity, the software is encapsulated in submodules. Each submodule has it's own SDK, service and restservice.
As crawling is one of the core features of the software, this module has the ability to use plugins and provides an interface inside it's SDK.

2.1 Functional Requirements

The PFC needs to know which repositories are available by Gitlab.
The plugin has to be able to detect them. The context of archived and forked repositories can differ for every user, the user shall be able to skip them. Also commits that are pure merges need to be skipped, as they are not evaluated for the patch-flow. After the repositories are fetched, the FPC crawls the commits for each project.
The apter must be able to retrieve the patches. These patches need to be retrieved and sorted by their ascending creation time. Then the FPC knows exactly, which patches already haven been crawled and no duplicated work is done.
Each fetched patch also includes filechanges, that need to be fetched as well.

Each patch has one author and committer which are instances of the user entity. Mapping those users to each patch is an important step for successfully calculating the patch-flow.
Consequently each patch shall get automatic assigned to the according PFC representation of a person, for both - the author and the committer.
Patches that cannot get assigned automatically shall be listed and manually assignable in the webinterface.

As Gitlab provides more functionality, than a local git repository does, it available meta-data shall be used - like generating the organization structure from the provided meta-data.
To make the plugin extendable and allow other developers to work with the source-code regression tests in form of unit and integration-tests needed to be implemented. This shall also ensure that bugs in the Gitlab-API are covered.

2.2 Non functional Requirements

The most current Gitlab-API-version is version 4. All lower APIs have been dropped in previous versions of Gitlab. There is a note over GraphQL, which is as alternative query-language instead of REST, but due license issues, this is not released yet. Summa summarum the adapter only needs to support version 4 of the REST-API.
As the application is designed for inner source, which implies the usage of companies, the implementation needs to be able to scale up for big projects. It also needs to be able to run on systems with moderate memory and processor power. For debugging and usability reasons the user should always be able to see the current state of the CrawlRun. Proper log output should always be provided and be configuratable due the logging level.

2.3 Evaluation

As this is an engineering thesis on an active developed software, each contribution is considered as accepted when a merge request, that implements this features was merged into the stable develop branch.
As Mr. Capraro is the founder and main author of the PFC, his opinion is considered as expert opinion in the full scope of this thesis.
Therefore an accepted merge grants sufficient code quality, out of the two-man rule in combination with stylecheckers and automatic tests of the CI-pipeline.
A merge request (MR) was also used to discuss different approaches to certain problems. They also document brainstorming, solutions beside from the classical code review.
But as a MR is a pure code contribution, which does not grant the implementation of the full extend of the specification. A special appointment with Mr. Capraro as supervisor was made. At this appointment each requirement was tested for its fulfillment on the internal Gitlab server of the RRZE.

The combination of the already reviewed code, an open-source oriented mindset and the evaluation of a CrawlRun in productive real-world environment makes the conclusion of this thesis.

3 Implementation

The implementation section is divided into two parts. First the design of the SDK, which provides the pure functionality library and afterwards the construction of the plugin, which implements the actual features.

3.1 Design of the SDK

The adapter itself is written in Java8. As we are interacting with an REST interface, the choice of the HTTP-library is crucial. As HTTP/REST client, Retrofit2 already was used broadly in the PFC codebase and provides strong static-typing and automatic parsing of the JSON-responses. This also fits very naturally in the development environment.
Retrofit abstracts all seven layers of the OSI model. Therefore it requires an interface, which holds the necessary data, like the used HTTP method, query parameters and service URI. This data is passed in method annotations as they appear in listing 3.1.
For transforming the JSON response into a POJO the GsonConverter is used.
The cost of this abstraction is, that all Object-Orientated-Principles are broken for the given interface. So a general contract can't be established and future Gitlab-API versions need a completely new interface.

An overview of the SDK design can bee seen in the UML of of appendix A. This will be further explained in a bottom-up approach.

3.1.1 GitlabService – Authorization and endpoints

The interface mentioned above is implemented in the GitlabService class. It provides the hardcoded endpoints and parameters for each API call.
To hide the boilerplate code for creating a Retrofit implementation of this class

the GitlabServiceFactory is used.

In the GitlabServiceFactory the AuthorizedHttpClientFactory is called.
As Retrofit itself relies on OkHttp to handle the networking, it is possible to hand in a custom OkHttpClient.
OkHttpClient allows to add so called 'Interceptors'. Interceptors are chained and are able to customize the raw HTTP-Requests.
In this place the private-api-key for Gitlab is added automatically to each request as query parameters. Adding this key to every request decreased the complexity for the GitlabService, as all authorization is now implicit handled and the service can focus on pure functionality.

```
@GET("projects/{id}/repository/commits")
Call<List<GitlabCommit>> getCommits(
    @Path("id") String _path,
    @Query("since") Date _since,
    @Query("until") Date _until,
    @Query("per_page") int _pageCount,
    @Query("page") int _page
);
```

Listing 3.1: GitlabService example method

3.1.2 GitlabClient - Fetching Data from Gitlab

The network layer and JSON to POJO mapping gets encapsulated in the GitlabClient class.
This class validates the global configuration file settings and delegates the GitlabService. The GitlabService class only holds methods which are Retrofit2.Call Objects. Such an object offers an execute method. In this method the network call, the JSON parsing and HTTP response code check happens, so all non deterministic exceptions get handled there. When a exception occurs, the call is repeated until a counter variable reaches a certain threshold. As this code is pure boilerplate it gets wrapped into the generic function, 'validateResponse'. This method gets a 'RetrofitRequestBuilder'-Interface. This interface only holds a 'fetch' function. This function wraps around the according GitlabService method call, by creating a closure.
Java interfaces with only one method can get called over a lambda-expression. This makes the GitlabClient methods one-liners, as one only holds the inline creation of the interface implementation, which is passed as parameter to the validateRespone method.

As shown in listing 3.2, the resulting code is, beside the type information, very similar to more expressive scripting languages like JavaScript or Python.

```
public List<GitlabCommit> getGitlabCommits(GitlabProject _proj, Date
    _since, Date _until, int _count, int _page) {
    return validateResponse(() ->
        service.getCommits(_proj.getPathWithNamespace(), _since,
        _until, _count, _page));
}
```

Listing 3.2: GitlabClient example method

3.1.3 GitlabPaginator – Utilization of the Gitlab-API

The Gitlab-API heavily relies on pagination. Every request returns a maximum of 20 elements per request, per default. This can be increased up to 100 elements by setting a query parameter.
Pagination on code level itself is pure boilerplate, but by fetching different elements various conditions arise. The class 'GitlabPaginator' uses therefore the static and generic function, 'paginate', which awaits the abstract class 'PaginationRequest' as parameter.
This class holds the abstract method 'fetchFromClient', with the current page-number, and the elements per page as parameters.
An implementation of the method expects the page-number and element count as parameter and delegates them, to their underlining GitlabClient method.
In order to be able to customize such a request, the virtual functions 'takeWhile' and 'filter' are implemented. They are inspired by the Java9 Stream Interface. The default implementation of those methods triggers a non-behavior. But PaginationRequest implementation can optionally override this methods, to customize request. Listing 3.3 shows, how this results in a clean way of creating customizable interfaces.

```
public static <T> List<T> paginate(PaginationRequest<T> _request) {
   List<T> addedObjs = new ArrayList<>();
   int page = FIRST_PAGE_INDEX;
   // Traverse trough pages
   while (true) {
      List<T> possibleAddObjs =
          _request.fetchFromClient(ELEMENTS_PER_PAGE, page);
      // Add every repo, that fit the constraints
      for (T checkObj : possibleAddObjs) {
         if (_request.takeWhile(checkObj)) {
            if (!_request.filter(checkObj)) {
               addedObjs.add(checkObj);
            }
         } else {
            return addedObjs;
         }
      }

      // Check for end of data stream
      if (possibleAddObjs.size() < ELEMENTS_PER_PAGE) {
         break;
      }
      page++;
   }
   return addedObjs;
}
```

Listing 3.3: Implementation of the paginate method

3.1.4 GitlabCommitIterator – Effectively fetching Patches

Fetching patches is a fundamental feature for the FPC. For this reason the 'ScmAdapter' interface exists. The contract phases that implementation should be able to fetch patches of a given project, in a given time range, sorted ascending by their creation timestamp.
To have a considerate memory consumption, the 'fetch' function returns an Iterator-Interface, which lazy fetches the patches, instead prefetching and creating one big queue with high memory consumption.

The Gitlab-API has the functionality to return commits in a given time range, using the query-parameters 'since' and 'until' but the patches are returned in a descending order.

Consequently we need to return the patches in a reverted order. As just reverting the list of all patches would break the idea behind the Iterator interface.
The solution was to determine the last page of all returned commits.
Gitlab sends the amount of pages for each request in the request-header field 'X-Total-Pages'. So before fetching actual commits, this field is evaluated and the pages are buffered in a reverse order.

Before returning a patch in the iterator, the commit diff needs to get fetched as well. This includes calling the according function in the GitlabClient class, with the accordingly is pagination.

3.1.5 Transformer – Enter the PFC world

The JSON responses from the Gitlab-API are transformed into POJOs by the GsonConverter internally. To be able to do this, the according Java attributes are annotated with the JSON name fields. This includes naming convention from JSON snake_case, to Java CamelCase.
The POJO do not fully represent the received JSON. As there was a lot of data, that could not be represented by the PFC, or irrelevant meta-data.

Beside the classes being used as pure data-structures, there are some convenient functions, to give the internal values a more meaningful naming. Like the 'isBinary' function of 'GitlabCommitDiff'. This methods checks if Gitlab prefixed the an internal field, with an undocumented magic value.
In table 3.1 is an overview of the retrieved Gitlab classes, and their according representation in the PFC.

Gitlab-Entity	**PFC-Entity**
Project	Repository, InnerSourceProject
Commit, PFC-Repository	Patch
Commit, CommitDiff	FileChange
User	Person

Table 3.1: Overview of transformation enitities

Transformation of GitlabRepositories

A repository has no one to one representation in the PFC.
The 'Repository ' class is used to hold the URL of the GitlabRepository. As this class only is used for internal data management and is later required for actual data-fetching.
The 'InnerSourceProject' instead gets the project name, with and without the projects name-space and association to the created repository object.

Transforming GitlabCommits

To transform a GitlabCommit into a PFC-Patch, the associated PFC-Repository is necessary for retraining their internal coupling.
As a GitlabCommits contains a list of GitlabCommitDiffs, which are represented as FileChanges in the FPC, and those are transformed as well. Therefore the list gets iterated and added into the resulting patch.
The pseudonym for author and committer is a concatenated string, formatted with a git naming convention: "name >email<¿"

To transform into a FileChange, there is the 'DiffProcessor' class, already in the SDK package. This class parses the changed and deleted lines out of the diff-string. But as it was written for the local file-system-git which does slightly differ in their file format from the Gitlab-diffString. To make this compatible it is necessary to prefix a magic value.

The code of the Transformer is very straight forward, but very criticial as they appear in appendix B.

3.1.6 GitlabAdapter- Setting it all together

To encapsulate the Gitlab specific objects and details from the plugin module, the GitlabAdapter was created. ScmAdapter-Interface and delegates the already mentioned 'GitlabClient', 'GitlabPaginator', 'GitlabCommitIterator' and 'Transformer'. It is also used to read in the user configuration and creates the authorized client.
As shown in listing 3.4 business method of this class returns the result a 'Gitlab-Client' API-call, wrapped into the 'GitlabPaginator' and finalized by the 'Transformer'.

```
protected List<GitlabProject> loadProjectsFrom(boolean _archived, Date
    _from) {
   // Traverse trough pages until a old repo appears
   return GitlabPaginator.paginate(new
       GitlabPaginator.PaginationRequest<GitlabProject>() {
      @Override
      List<GitlabProject> fetchFromClient(int _pageContentSize, int
          _page) {
         LogUtil.info(getClass(), "Loading projects page {}", _page);
         List<GitlabProject> projs =
             client.getGitlabProjects(_archived, _pageContentSize,
             _page);
         LogUtil.info(getClass(), "Loaded projects page {} - count {}",
             _page, projs.size());
         return projs;
      }

      @Override
      boolean takeWhile(GitlabProject _checkObj) {
         return _checkObj.getDate().after(_from);
      }

      @Override
      boolean filter(GitlabProject _checkObj) {
         if (skipForks && _checkObj.isFork()) {
            LogUtil.info(getClass(), "Skipping fork: " +
                _checkObj.getPathWithNamespace());
            return true;
         }
         return false;
      }
   });
}
```

Listing 3.4: GitlabAdapter example method

3.2 Plugin Implementation

As the SDK only provides functionality, but doesn't implement any innovation the 'GitlabDemoPlugin' is implemented. This class extends the 'AbstractPlugin' class and dues its contract implements the return of 'Pre/PostSteps' and 'PatchProcessors'. Each of this steps is called sequentially at the according time of a 'CrawlRun'.
A 'Pre/PostStep' does not have a particular contract to fulfill, beside of the configuration the PFC for the given plugin. A 'PatchProcessor' get's every new fetched patch, and is allowed to modify or even filter this patch.
Also the already mentioned ScmAdapter needs to be provided.
This thesis implements the following PFC Plugin interfaces:

PreProcessors:

- AddRepositoryPreStep:
  Fetching, transforming and saving the repositories

PatchProcessors:

- AttachInnerSourceProjectRepository:
  Adding the according ISP to the patch
- ResolvePseudonymProcessor:
  Resolving already mapped author-pseudonyms
- GitlabAddAuthorProcessor:
  Resolving new author-pseudonyms

ScmAdapter:

- GitlabAdapter:
  Fetching, transforming and saving the patches with their according filechanges

3.2.1 Configuration

The plugin requires the following configuration by the user:

- baseUrl:
  The URL to the Gitlab-API, eg. www.gitlab.example.com/
- apiToken:
  A generated 'Access token' with the 'api-scope' as requirement. This is required as Gitlab does not provide any functionality to login with regular credentials

- skipForks:
  Flag if the Crawler shall ignore forked repositories
- skipArchived:
  Flag if the Crawler shall ignore archived repositories

3.2.2 AddRepositoryPreStep – Setting up repositories

Before the PFC can fetch any patch, is has to know which repositories are available. The 'AddRepositoriesPreStep' provides this functionality. It determines the date of the last successful CrawlRun and request all Repositories and InnerSourceProjects, created since this date from the GitlabClient. The retrieved Objects are getting saved in the database.

3.2.3 GitlabAdapter - Fetching the Patches

For fetching the patches, the PFC uses the ScmAdapter-interface implemented by the GitlabAdapter. The PFC iterates over all repositories, and fetches all patches, between the time of the CrawlRun and the last successfully saved patch in the database.

3.2.4 ResolvePseudonymProcessor – Reuse pseudonyms

Gitlab does not provide any functionality to assign an Patch to an author by default. Gitlab, unlike other software forges e.g Gitlab, does not return the id of the author or the committer. While this makes partly sense for the author, who does not necessarily need to be a user of the Gitlab-Instance, the committer definitely has to be one. It is also not possible to list commits by a user. A according patch was not merged into Gitlab (#12760), even it was demanded by the community.

This makes automatic mapping a non trivial thing and the quality of the outcome heavily relies on the proper configuration of the pseudonyms, which are locally configured on the contributors machines.

As the patches get processed, the 'ResolvePseudonymProcessor' makes an query into the PFC database, if there is any patch already having a mapped person to the current pseudonym. As a pseudonym can be used as authorPseudonym and committerPseudonym, two database queries are required. The two lists get

unioned and then evaluated. This allows automatic mapping of already manual mapped pseudonym and increases performance for automatic mapped ones, as there isn't an additional API-Call.

As this solution is not specific for Gitlab, but for the PFC, is part of the pseudonymresolver package and not the one for Gitlab.

3.2.5 GitlabAddAuthorProcessor – Map new Users

When there isn't a already mapped pseudonym and therefore the ResolvePseudonymProcessor mapping fails, the 'GitlabAddAuthorProcessor' tries do determine the Person with the Gitlab-API.

Consequently the email is extracted from the pseudonym. This is particularly easy, as the plugin controls the format of the names and just can use a regular expression to extract it.
The retrieved email is queried in the Gitlab search endpoint. This search endpoint provides a fuzzy search. But when sending a valid email, Gitlab thankfully only returns matching users entities. This is not documented, but makes it way easier, as the plugin does not have to evaluate the return type first.

If only one user is returned, the 'PatchProcessor' assumes, that Gitlab did the email matching successfully and matches the user to the patch. Every lookup email is cached, so that duplicate emails are not queried multiple times.

3.3 Manual mapping pseudonyms

The PFC has an webinterface. This webinterface is written with Angular6 and typescript as frontend with a jersey/glassfish REST-backend. In order to being able to map patches manually to persons, the according components in the datamanger-modules were implemented.

3.3.1 Webclient - Provide functionality to the user

The used design language is the material design by Google. This fits very nicely into the angular application, as google provides presets out of the box.
The feature to manually map users to certain pseudonyms are assigned to the

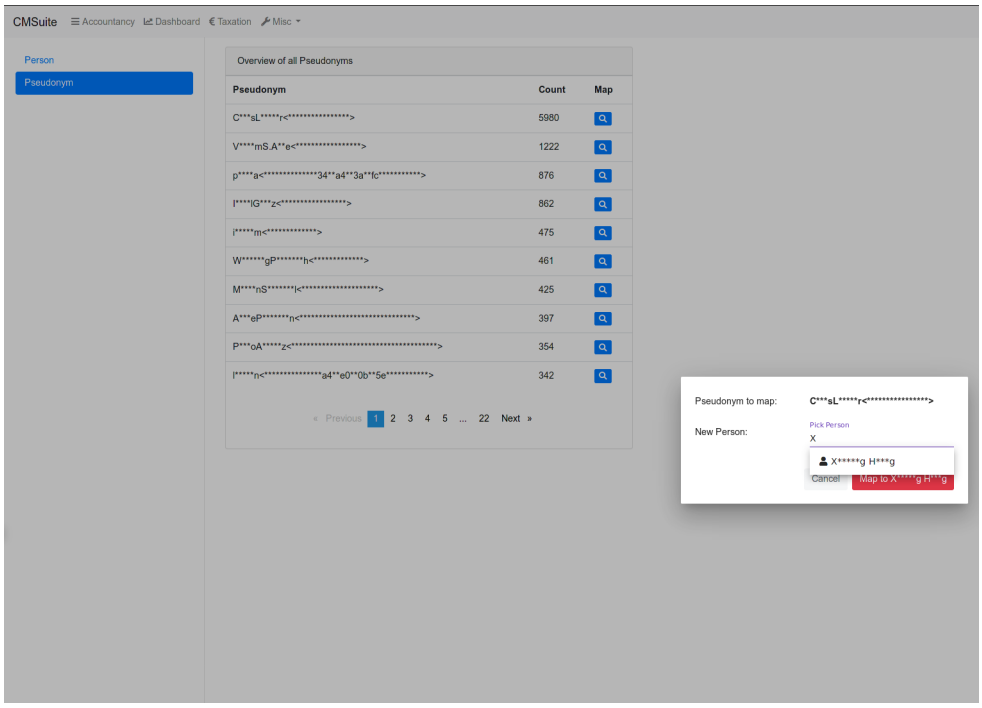


Figure 3.1: Screenshot of the pseudonym-table with active picker-modal

'Data Managment' features. Placed a option in the dropdown menu of the 'Misc' category, which is placed in the main toolbar.
The component shows a paginatable table. The table columns are the pseudonym, the amount of unmapped patches and a button to map the according entry, as shown in figure 3.1.
The entries a sorted by descending by the patch-count.
Each time the table got shown a GET-Call to the backend is made, which returns the list of all unmapped pseudonyms, and will be explained in detail later.

Clicking the 'map' button opens a modal. This modal is showing the pseudonym and a inputfield, beside the obligatory submit and cancel buttons. When typing in to the inputfield, an autocompletion get's triggered and shows all persons, matching the name or email. This autocompletion is also done via a GET-Call. This allows to access to the whole Person-Object including the person id, which is mandatory for the further mapping.
When a valid person is chosen, the submit-button gets enabled. The communication with the backend happens in a PUT-Call, with the pseudonym as path parameter and the to mapped person as JSON in the body.

Mapping an already mapped pseudonym to a different Person, would create un-

wanted complexity in the backend. Therefore once a pseudonym gets mapped, the button is disabled, after refreshing the site the pseudonym is mapped in the backend and no longer get's returned by the backend.

3.3.2 The backend logic

To communicate with the frontend the backend got the 'PatchResource', which holds the according endpoints and delegates the PatchService to execute the business methods. For interaction with the database the PatchDao got extended.

3.3.3 Getting unmapped pseudonyms

The Resource is able to return all unmapped pseudonyms in the 'getUnmappedPseudonyms' method. This endpoints holds the boolean variable unmapped as query paramenter. As currently only unmapped pseudonyms are required, a 'UnsupportedOperationException' is thrown, when the boolean is not true.

The delegated service directly returns the query. As DTO class the 'PseudonymCount', which only holds the pseudonym string and the count of unmapped patches was created. As this class isn't an Hibernate-Entity, the default POJO transformer couldn't get used and the Hibernate 'aliasToBean'-transformer. When the 'SQLQuery' gets this transformer instead of an enity the class attributes get assigend according to the query columns names. Therefore DbReader was edited and now sets the transformer depending on the annotations of the given class.

The key was to union all patches from pseudonyms without mapped author or committer. This are two intersecting sets. To guarantee the best performance and keeping the complexity low, the distinction and sorting shall be done by the database. Label 3.5 shows the resulting query. It unions two subqueries, one for the patches, without authors, and once for ones without committers. The union is done due the use of a SQL projection. So the result is distinct set, grouped by names and sorted by their patch count. All done by the database.

```
SELECT pseudonym, count(id) as count
FROM (
    SELECT p1.author_pseudonym AS pseudonym, p1.id as id
    FROM patch AS p1
    WHERE author_id IS NULL

    UNION DISTINCT

    SELECT p2.committer_pseudonym AS pseudonym, p2.id as id
    FROM patch AS p2
    WHERE committer_id IS NULL
 ) AS q
GROUP BY pseudonym
ORDER BY count DESC, pseudonym DESC
```

Listing 3.5: Querying unmapped pseudonyms

3.3.4 Manually mapping a pseudonym

The second method of the class is the 'mapPsedonymToPerson' method. It expects the pseudonym to map as path parameter and gets the according person as body parameter in the PUT request. Jersey is implicit handling the conversion from JSON to POJO.

The 'mapPsedonymToPerson' method is the endpoint for mapping a pseudonym String to a according person object. The ressource asserts that there is a valid person object, by asserting a valid id and then is updating all patches. This is done by first querying and updating all patches with the matching author-pseudonyms and then the same procedure with the committer-pseudonym.
To improve performance, batch updating got implemented in the 'DbWriter' class.

3.3.5 Creating the organizational structure

Gitlab has the groups and sub-group function. Here the users can get gathered. But the default use-case is to access control. As it is convenient to manage whole groups, instead of single persons. Also some addition namespace configuration is possible. Using this for organizational structuring seems be irrelevant in practice. Conceivable target groups would be small Start-ups, which do not rely on any command chain, but as this is not the target group of the cmsuite, this idea is dropped.

3.3.6 Logical inference for mapping Authors to Patches

As every not mappable pseudonym has at least one author, it is possible to automatically map those patches, if and only if one author added patches. It is save to assure that this wont be the case for long lived projects. But with properly configurated projects and an automatically executed CrawlRun within the CI-Pipeline, this assumption would actually hold, as usually only one developer adds new Patches to the project per merge request. So this developer could misconfigure his local git and the cmsuite will still be able to assign the according patches. Due the 'ResolvePseudonymProcessor' it would even automatically map this pseudonym to the right user.
But as this is very unlikely in practice, the idea was dropped. Although it can get implemented as 'PatchProcessor' in the future.

3.4 Testing

As core Requirement for this thesis, a well rounded test strategy is demanded. This includes Unit tests, as well as Integration tests. The resulting testing suit, also needs to be executable in the CI pipeline of the project.

3.4.1 Setting up an testing environment

The cornerstone for the suite is a selfhosted Gitlab instance. As using a existing Gitlab, wouldn't be extendable for various Gitlab versions. Also modifying the test data, would be problematic, when two pipelines are executed in parallel.

To be deployable in the CI pipeline the instance is hosted via a docker container. The existing and already configured dockerfile from the official GitlabCE repository, couldn't be extended, because there is a docker 'RUN' command, which is restating the server in an endless loop, when executing the script.
Therefore the existing code was used as base for the new dockerfile and is equivalent of an Gitlab installation on a vanilla Ubuntu 14.04.

After installation only one bash command is necessary to set up, configure and run then Gitlab server: 'gitlab-ctl reconfigure'. This is setting up the all necessary data structures, reading in the configuration files and starting the server. But this command takes up to five minutes and is very I/O intensive, which makes it impractical to run it inside the CI-Pipeline.
The command 'gitlab-ctl start' instead only starts the server and all services,

but requires an already exectued 'reconfigure' command. The command itself is finished in a few seconds. The final up time for the services is around 45 seconds, depending on the used hardware and Gitlab version.
As docker easily allows the usage of images, the final docker-image, already has executed the expensive 'gitlab-ctl configure' command and only needs to run the 'start' command.

Beside the installation configuration was done. Mainly used to change the default port from 80 to 10080, as that is the redirected also docker port. This makes it easy to use the same scripts, for development and deployment, as the endpoints don't change. The root password also got changed.

Filling the Gitlab instance with mock data, was the main issue of this task.
Gitlab does not allow login in via a username, password scheme. A API-Key needs to be created inside the Webinterface.
Just creating a new Gitlab instance, every time the mock-date changes, would finally change the private API-Key. This especially inconvenient, as the key is hardcoded in the config files of the cmsuite project. As the docker-images are a only subproject of the cmsuite, changing the API key would produce an additional MR in the main project. Also other developer would have to keep their project always up to date. Otherwise failing pipelines, on parts their are not working on would yield and restrict their workflow.

3.4.2 Creation of an Docker backup

There is a script inside the dockefile, that creates a backup. As it only differs two lines from the original dockefile, it is in the script but commented out.
This script creates the backup and copies it to a specific folder. By using the docker-host command it is possible to transfer the backup file to the host-machine.

Additionally the script creates the API-Key and persists it. This is done via parsing the fetched HTML and faking the according HTTP request. The script is written in python3 and make use of the request library.
The use-case for this workflow is updating gitlab, as gitlab backups are only compatible with the current version. With the usage of the script, the developer now has no need to login inside of the webinterface, creating and saving the key by hand.

Down below are the necessary steps for the developer, that creates a new backup file:

- Setting the gitlab version in the file
- Comment in/out the according lines in the dockerfile
- Configurate the docker-compose file in the utils folder, to point to a existing folder on the host machine
- Start the modified container
- Copy the created files inside the the assets/backup folder
- Adjust the permission of the files

3.4.3 Inserting testing data

There are two types of data, which are necessary to test for this thesis. Gitlab specific data and git only data.
Gitlab data like users, groups and project, where git data is only the version control.

Creating git related data is trivial, even the automatic creation via bash.
For the Gitlab data-structures an other python3 script was written. It relies on a Gitlab-API wrapper library.
Due a implemention of a shim code layer, it is possible to create users, groups, project and forks in a single python line.

To populate the database, the RUN command of the docker-file creates a new Gitlab instance, restores the backup, with the API-token, executes the Gitlab-populate script and finally pushes the just created git project to Gitlab.

The final Gitlab data is the following:

- Creation of 15 Users
- Creation of three projects
- Creation of a branch of two projects
- Fork of a Project
- Archiving of a project

And the git data contains these changes:

- Creation of a file
- Add a line to a file-delta
- Remove a line to a file-delta

- Renamed a file
- Changing the git_author
- Creation of a binary
- Patch of the created binary
- Changing of the git_committer_email

3.4.4 Integration in the Pipeline

The pipeline is using a global docker-compose file, where a depending services get started. The Gitlab image was added there.
This docker kickoff is done in parallel with the compilation step. This is the best possible minimization for the uptime delay.
But to ensure the test-suite is fully booted, an watchdog is implemented, which checks if the services are already up. This is done by just curling the Gitlab landingpage and checking the return code. Wrapped up by some code, to avoid an endless loop.
The integration tests are executed by maven in an subsequent step of the pipeline.

4 Evaluation

The final evaluation was done on the RRZE Gitlab. This instance is used, as it is public for students of the FAU and has a reasonable amount of data. Its data pool is small enough to still allows manually crosschecking the cmsuite database content, with the data of the Gitlab webinterface with a natural limit for the a CrawlRun runtime. But is also big enough to provide an meaningful stress test on the application.

4.1 Performance and testing system

The application was tested on a regular notebook with an i5-5200U and 16GB of RAM, with an up-to date Fedora 28. The machine always was responsive and had an average CPU load of 40%. 1,05GB of RAM was used when running the application in IntelliJ Ultimate 2018.1.
The average amount of fetched patches are around 20.000 per hour. With a runtime of 46 hours.

4.2 Database and validity

As descripted in the table below there are almost a million patches in 504 projects and 165 persons. The overall table count can be seen in table 4.1 and excerpts of the table contents can be found in the appendicies C-G.

The crawled data will only analysed withing a semantic context, as dermining the data quality would require all other patch-flow relevant data. Manually checking the count of fetched patches in the database with the count of the Gitlab webinterface was done. Hence the five biggest repositories and two randomly selected repositories were used. The comparison showed the equal amount of patches. Checking the content of the respectively last repository-patch, also showed equal data.

Tablename	Count
Crawlrun	1
Filechange	2725974
Innersourceproject	504
Innersourceprojectlink	0
Orgdimension	0
Orgelement	0
Orgelementttype	0
Orglink	0
Patch	909539
Person	165
Personlink	0
Repository	504
Resulthrow	0
Taxationpreference	0
Transformation	0
Transformationrun	0

Table 4.1: database table count

Checking critical Sections, like parsing user names and project namespaces was done manually. No anormal data was found.

As the CrawlRun only resolved 165 Persons, it is obvious, that the patch mapping was not very effective. The percentage of mapped patches was only 5%.
This is mainly as the biggest projects are open-source project mirrors. Mostly llvm variants and even one UNIX kernel. Those are not marked as forks in the RRZE-Gitlab, as they are inofficial mirror repositories - indepenent from the official software forge. Those projects hold commits, which are done by pseudonyms that can not be known by the RRZE-Gitlab instance - and are therefore not resolvable.

The detection is also highly depending on the local git configuration. As the instance is mostly used by students. And it is very unlikely that this target group properly sets commit name and email for their private and university projects. As this is the projects are mostly relevant in the internal university context.
In the CrawlRun on the research group, with Mr. Carpraro as supervisor, the mapping rate was 75%. This is significant higher, but will need the extra step of manual assignment, which was implemented in this thesis.

4.3 Logging and resilience

Produce traceable logfiles is part of the functional requirements of this thesis. The generated logfile has the highest logging level, which is info, and holds overall 3.678.850 lines.

As the info output is for debugging and progress-checking purposes is not further evaluated.
Appendix I is an summary of all warn-level logs, with the included tracelogs. This appendix shows five repositories, which couldn't get crawled properly.
The reason for this always was a readtimeout, which coulnd't get resolved, by just repeating the call. As the readtimeout already was increased to ten seconds, this ascribable to the lacking performance of the target server.
Due the functionality of the the cmsuite, those repositories would get crawled on the the next CrawlRun.

The error logs show 25 errors.
20 of those errors complainging over not determinable page size to fetch the commits. But accessing the repositories over the webinterface is also not possbile, as those repositories do not longer exists and seem to be only reachable over the Gitlab-API endpoint.
The other five errors, are the error messages, triggered by the read-timeout mentioned above.

4.4 Testing

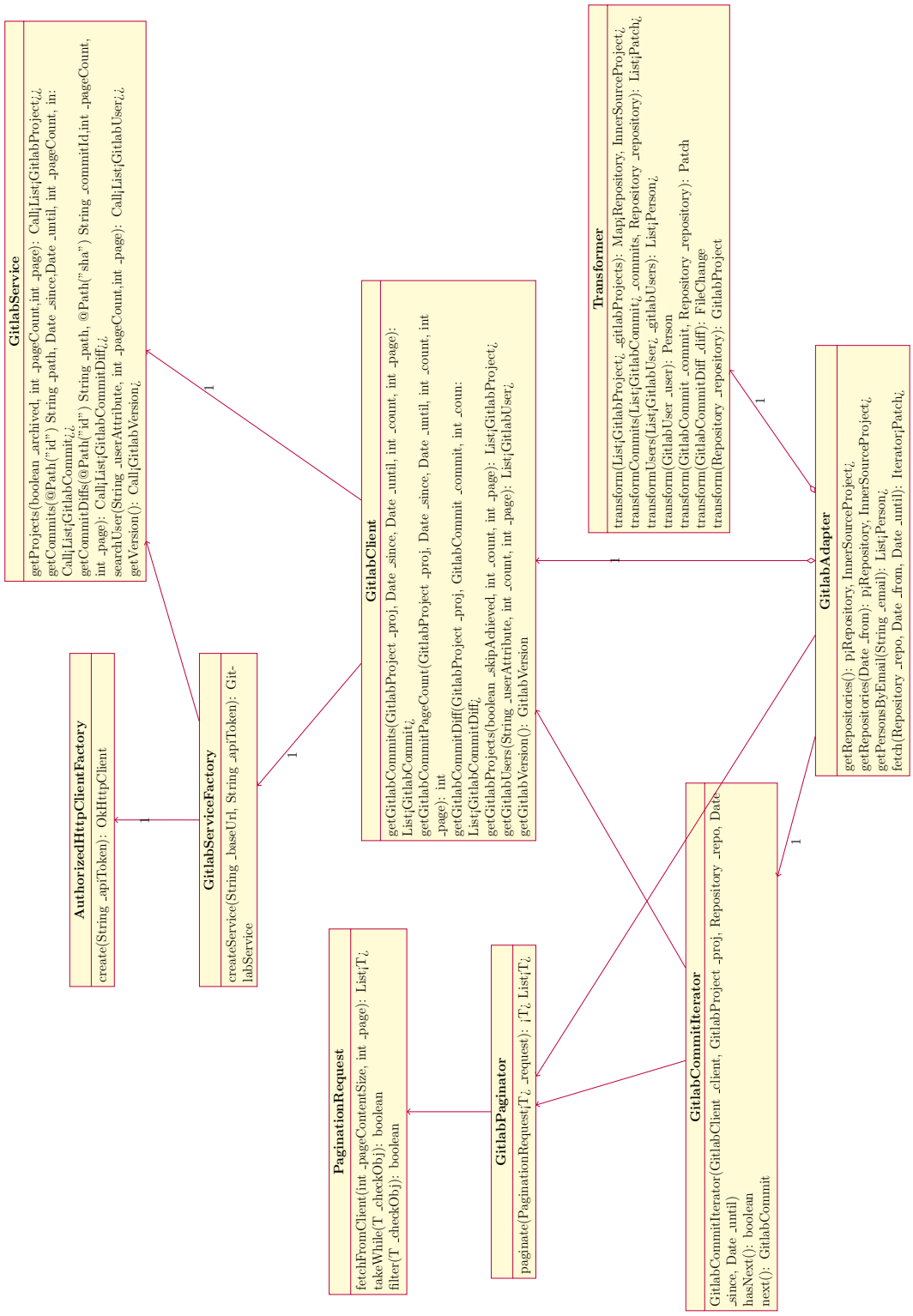
Unit tests only cover the Transformer and Gitlapdapter class.
While the coverage on the first one is 85% for all lines. The Adapter itself has 59%.
While testing the Transformer is trivial, testing the Adapter is focused on testing it's depending subclasses. Therefore the GitlabPaginator has a coverage of 94% and the CommitIterator has 90%.

Integration testing was the crux for this thesis. As the Gitlab instance is a double-edged sword. It guarantees stable mock-data but the up-time are long 45 seconds, every time the pipeline get's started. The parallization with the compilation step only weakens this delay. While this is still an I/O and time expensive part of the pipeline.

The integration test itself take ten seconds on normal hardware. The GitlabClient has a line coverage of 82% and all relevant Endpoints are tested properly. The coherend Factory classes are fully covered.

The coverage of both testing attempts have a line coverage of 71% and a method coverage of 78% on the total gitlab package. This isn't optimal. But as long as breaking API changed, like the so called bugfix of the archive flag, happen, the SDK has working and fully tested core features. An other author, which is extending the functionality to an further scope of this thesis can rely on the tested core features, like fetching, response-pagination and transforming.

Appendix A UML Gitlab



Appendix B Transformer code

```
public Patch transform(GitlabCommit _commit, Repository _repository) {
   Patch patch = Patch.create();

   // Set default values
   patch = patch.setRepository(_repository);
   patch = patch.setToken(_commit.getId());
   patch =
       patch.setCommitterPseudonym(_commit.getCommiterName().trim() + "
       <" + _commit.getCommiterEmail() + ">");
   patch = patch.setAuthorPseudonym(_commit.getAuthorName().trim() + "
       <" + _commit.getAuthorEmail() + ">");
   patch = patch.setCommitDate(_commit.getCommitedDate());
   patch = patch.setCommitMessage(_commit.getMessage());

   // Set fileChanges
   Set<FileChange> changes = new
       HashSet<>(_commit.getGitLabCommitsDiffs().size());
   for (GitlabCommitDiff diff : _commit.getGitLabCommitsDiffs()) {
      FileChange change = transform(diff);
      changes.add(change);
   }
   patch = patch.setFileChanges(changes);

   return patch;
}
```

Appendix C Excerpt Person

Id	Token	Email	Firstname	Lastname
44	1074	j****.**.*****r@fau.de	J***s	M****r
76	501	d****.********n@fau.de	D****l	Z*******n
153	950	m****.****l@fau.de	M***m	O*****l
85	771	d****.********n@fau.de	D***d	S********n
15	381	a******.*****l@fau.de	A******s	H***l
11	16	p******.****t@fau.de	p******0	
14	66	m***.****n@fau.de	M***c	A****n
66	388	h*****.****m@fau.de	H*****d	B**m
7	1099	t*****.******r@fau.de	T****s	W****r
146	440	m******.*******f@fau.de	M****l	P******f
130	1079	f****.****d@fau.de	F****z	S****d
54	206	t******.********r@fau.de	F*******u	
106	596	w****.******h@fau.de	W****r	S******h
144	1377	x******.****g@fau.de	X*****g	H***g
65	1521	j****.******l@fau.de	J***b	D****l

Appendix D Excerpt InnerSourceProject

Id	Token	Name	Root	Repository
300	StuveFAU/satzung-securitykosten	satzung-securitykosten (StuveFAU)	/	300
308	by92gyfi/kOpt	kOpt (by92gyfi)	/	308
358	MobCG/Assignment1	Assignment1 (MobCG)	/	358
2	ik15ydit/fauiwg-foxtemp	fauiwg-foxtemp (ik15ydit)	/	2
419	zy29tela/rechnerkommunikation	Rechnerkommunikation (zy29tela)	/	419
465	ex12uqod/stubsmi	stubsmi (ex12uqod)	/	465
9	fsv-tech/ak-ese-heft-downloads	ak-ese-heft-downloads (fsv-tech)	/	9
181	faumachine/faumachine	faumachine (faumachine)	/	181
85	Matombo/libLessPain	libLessPain (Matombo)	/	85
234	ex12uqod/regenampel	Regenampel (ex12uqod)	/	234
174	DerRoteBaron/user_scripts	user_scripts (DerRoteBaron)	/	174
183	i4/passt/git-bisect-demo	git-bisect-demo (i4)	/	183
1	i4llvm/i4instrument	i4instrument (i4llvm)	/	1
385	icipguru/cip-foxtemp	cip-foxtemp (icipguru)	/	385
298	icipguru/pcidentd	pcidentd (icipguru)	/	298

Appendix E Excerpt Filechange

id	Filename	Path	Action	Del.	Ins.	Patch
2166413	variables.c	/test/PCH/	MODIFY	1	0	728870
615098	ClientReconnectTest .java	/src/java/test/org/ apache/zookeeper/	ADD	78	0	219077
522759	irgen.c	/test/Modules/	MODIFY	1	0	184343
480782	SemaDecl.cpp	/lib/Sema/	MODIFY	1	0	168668
2629289	linkage-name.ll	/test/DebugInfo/X86/	MODIFY	2	2	883176
502180	cxx-decls-imported.h	/test/Modules/Inputs/	MODIFY	3	0	177134
2358422	x86-64-arg.ll	/test/CodeGen/X86/	ADD	16	0	793893
1182766	SortIncludesTest.cpp	/unittests/Format/	MODIFY	13	0	431238
1261697	MCAsmInfo.h	/include/llvm/MC/	MODIFY	0	7	458570
911453	RegClass.cpp	/lib/Target/SparcV9/ RegAlloc/	MODIFY	11	11	335573

Appendix F Excerpt Patch

Id	Author	Commit Message	Author	Committer	ISP
672762	L**s G*******t <l***.**********t@fau.de>	Headings	115	115	431
113683	v*****h <v*****h>	Next step...	NULL	NULL	181
5241	T********n W*****n <edu@t*********-**********n.de>	Add Draft-Watermark	NULL	NULL	82
88783	v*****h <v*****h>	Cleanup step.	NULL	NULL	181
38591	<s*****@psp-at1.ntp.org>	merge cleanup	NULL	NULL	145
77	H********** <e**.*r.******@fau.de>	add ws 1617	3	3	10
97258	v*****h <v*****h>	Cleanup step.	NULL	NULL	181
107743	v*****h <v*****h>	Next cleanup step.	NULL	NULL	181
34403	<s*****@psp-deb1.ntp.org>	nits	NULL	NULL	145
102108	v*****h <v*****h>	Work in progress...	NULL	NULL	181
676540	A********r W*********n <a**@a**.name>	git drueber	NULL	NULL	446
275100	D****d C*****g <d*********g99@gmail.com>	Reversed removal	NULL	NULL	376
39429	<s*****@deacon.udel.edu>	NTP_4_3_83	NULL	NULL	145
39793	<s*****@psp-deb1.ntp.org>	updates	NULL	NULL	145
121109	v*****h <v*****h>	Next tiny step...	NULL	NULL	181

Appendix G Excerpt Repository

Id	Crawlrun	Location	Youngest Patch
436	1	https://gitlab.cs.fau.de/ze26zefo/eva-bookmarklet.git	674061
278	1	https://gitlab.cs.fau.de/etXzat/raspi-power-monitor-hat.git	207150
58	1	https://gitlab.cs.fau.de/is/templates/poster/fau-win-poster-template-latex.git	3398
261	1	https://gitlab.cs.fau.de/faumachine/faucc.git	202793
5	1	https://gitlab.cs.fau.de/iw18ejyg/bashrc.git	28
372	1	https://gitlab.cs.fau.de/nafiqft/data.git	269239
382	1	https://gitlab.cs.fau.de/i4/kss-tablegen.git	280389
146	1	https://gitlab.cs.fau.de/enos/enos.git	15977
207	1	https://gitlab.cs.fau.de/icipguru/cip-gitlab-wiki.git	
192	1	https://gitlab.cs.fau.de/thinegen/faui2k17-web.git	53700
399	1	https://gitlab.cs.fau.de/of82ecuq/gekritzel.git	329857
460	1	https://gitlab.cs.fau.de/iw18ejyg/webgps.git	713914
290	1	https://gitlab.cs.fau.de/neo/neo-lib.git	214611
191	1	https://gitlab.cs.fau.de/dosek-verification/dosek.git	53556
286	1	https://gitlab.cs.fau.de/bi40resu/cuba-corp-tests.git	209311

Appendix H API-Key retrievemnt

```
    def _request_api_token(session: requests.Session):
        """Performs an api_token creation and returns that token"""
        # Open to the private token site
        req = session.get(TOKEN_URL)
        if req.status_code != 200:
            die("Couldn't make GET on:", TOKEN_URL)

        # Parse an extra security token, which is baked in the html
        authenticity_token = KeyCreatorParser().get_token(req)

        # Simulate key-creation with an POST-request
        data = [
            # Needs to be a list with tuples, because of multiple
                identical keys
            ('authenticity_token', authenticity_token),
            ('personal_access_token[name]', 'AUTOMATIC_TOKEN'),
            ('personal_access_token[expires_at]', ''),
            ('personal_access_token[scopes][]', 'api'),
            ('personal_access_token[scopes][]', 'read_user'),
            ('personal_access_token[scopes][]', 'sudo')
        ]
        req = session.post(TOKEN_URL, data=data)
        if req.status_code != 200:
            die("Failed POST on: " + TOKEN_URL)

        # Returned website contains the fresh api_token
        return KeyRetrieverParser().get_token(req)
```

Appendix I Warn Log

```
[pool-2-thread-2] WARN
   de.fau.cmsuite.pfcrawler.sdk.gitlab.client.GitlabClient - Exception
   happened too often!
java.net.SocketTimeoutException: Read timed out
[pool-2-thread-2] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 40 at
   https://gitlab.cs.fau.de/zi39firu/diy-ss18-gruppe5-projekt.git
java.lang.IllegalStateException: Rethrowing:
java.net.SocketTimeoutException: Read timed out
--
[pool-2-thread-1] WARN
   de.fau.cmsuite.pfcrawler.sdk.gitlab.client.GitlabClient - Exception
   happened too often!
java.net.SocketTimeoutException: Read timed out
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 133 at
   https://gitlab.cs.fau.de/sithkarm/litmus-rt.git
java.lang.IllegalStateException: Rethrowing:
java.net.SocketTimeoutException: Read timed out
--
[pool-2-thread-1] WARN
   de.fau.cmsuite.pfcrawler.sdk.gitlab.client.GitlabClient - Exception
   happened too often!
java.net.SocketTimeoutException: timeout
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 138 at
   https://gitlab.cs.fau.de/frececroka/kalah.git
java.lang.IllegalStateException: Rethrowing:
java.net.SocketTimeoutException: timeout
--
[pool-2-thread-3] WARN
   de.fau.cmsuite.pfcrawler.sdk.gitlab.client.GitlabClient - Exception
   happened too often!
java.net.SocketTimeoutException: Read timed out
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 156 at
   https://gitlab.cs.fau.de/me61sewa/passt-mac.git
java.lang.IllegalStateException: Rethrowing:
java.net.SocketTimeoutException: Read timed out
```

```
--
[pool-2-thread-1] WARN
   de.fau.cmsuite.pfcrawler.sdk.gitlab.client.GitlabClient - Exception
   happened too often!
java.net.SocketTimeoutException: timeout
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 187 at
   https://gitlab.cs.fau.de/ob61ujeh/cyclegan-tensorflow.git
java.lang.IllegalStateException: Rethrowing:
java.net.SocketTimeoutException: timeout
```

Appendix J Error Log

```
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/al26yjiw/wedecide.git. PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/arw/hello.git.
   PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/efe/mmda.git.
   PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/energy/empya-akka.git. PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/energy/seep.git.
   PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/fsv-tech/logo.git.
   PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/icipguru/cip-gitlab-wiki.git. PageCount
   was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/lucareeb/KalahGUI.git. PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/mpfsi/protokollsystem.git. PageCount was:
   -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/vy37bypi/InversesPendel.git. PageCount
   was: -1
```

```
[pool-2-thread-4] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/gene/cart.git.
   PageCount was: -1
[pool-2-thread-4] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/jonglage/wissen.git. PageCount was: -1
[pool-2-thread-4] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/Visen/bs.git.
   PageCount was: -1
[pool-2-thread-2] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/kissen/passt-faunotify.git. PageCount was:
   -1
[pool-2-thread-2] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/mpfsi/klopapier.git. PageCount was: -1
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/az48uhaj/gameoflife.git. PageCount was: -1
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/el79irih/akss2015\_multics.git. PageCount
   was: -1
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/ro77zini/passt\_p.git. PageCount was: -1
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for https://gitlab.cs.fau.de/Visen/I4-EZS.git.
   PageCount was: -1
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.sdk.gitlab.GitlabCommitIterator - Couldn't
   determine page size for
   https://gitlab.cs.fau.de/zo27loxo/fagerte.git. PageCount was: -1
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 133 at
```

```
   https://gitlab.cs.fau.de/sithkarm/litmus-rt.git
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 138 at
   https://gitlab.cs.fau.de/frececroka/kalah.git
[pool-2-thread-1] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 187 at
   https://gitlab.cs.fau.de/ob61ujeh/cyclegan-tensorflow.git
[pool-2-thread-2] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 40 at
   https://gitlab.cs.fau.de/zi39firu/diy-ss18-gruppe5-projekt.git
[pool-2-thread-3] ERROR
   de.fau.cmsuite.pfcrawler.service.crawlengine.CrawlJob - An error
   occured crawling repo 156 at
   https://gitlab.cs.fau.de/me61sewa/passt-mac.git
```

References

Capraro, M., Dorner, M. & Riehle, D. (2018, May 28–29). The Patch-Flow Method for Measuring Inner Source Collaboration, 2–3. doi:10.1145/3196398.3196417

Capraro, M. & Riehle, D. (2017). Inner Source Definition, Benefits, and Challenges. *ACM Computing Surveys*, *49*, 8–14. doi:10.1145/2856821