UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Vladyslav Bondarenko

OpenSZZ - Evaluation and Improvement

Master's Thesis (30 ECTS)

Supervisor: Dietmar Pfahl, PhD

Tartu 2021

OpenSZZ - Evaluation and Improvement

Abstract:
SZZ is an algorithm proposed by Śliwerski, Zimmerman and Zeller to identify bug-introducing changes in software repositories. The algorithm consists of two parts. The first part is the identification of bug-fixing changes relying on information in an issue tracker. The second part is the identification of bug-introducing changes based on bug-fixing changes and relying on the annotation/blame feature of a version control system.
A few open-source implementations of the SZZ algorithm were proposed, and OpenSZZ is one of them. Although the SZZ algorithm has limitations, and some of them are already overcome, OpenSZZ implements the basic version of this algorithm. On the other hand, unlike other SZZ implementations, OpenSZZ is a cloud-native web application and implements both parts of the algorithm. Therefore, we improve OpenSZZ and propose an improved version of SZZ on top of it.
We found that OpenSZZ has issues with its implementation and provides incorrect results. In this thesis, we fixed found issues with OpenSZZ and improved it with features proposed in other SZZ implementations. In addition, we added an option to use information from issues of an issue tracker to identify bug-introducing changes. Finally, we added an option to analyze repositories without relying on an issue tracker.

Keywords:
SZZ algorithm, OpenSZZ, bug-introducing change, bug-fixing change

CERCS: P170 Computer science, numerical analysis, systems, control

OpenSZZ - Hindamine ja täiustamine

Lühikokkuvõte:
SZZ on kaheosaline algoritm, mille töötasid välja Śliwerski, Zimmerman ja Zeller, et tuvastada tarkvara repositooriumides vigu tekitavaid muudatusi. Algoritmi esimene osa on veahaldussüsteemi informatsiooni põhjal vigu parandavate muudatuste tuvastamine. Algoritmi teine osa on vigu tekitavate muudatuste tuvastamine, kasutades selleks teavet vigu parandavatest muudatustest ning tuginedes versioonihaldussüsteemi annotatsioonidele.
Eksisteerivad mõned avatud lähtekoodiga vabavaralised SZZ algoritmi implementatsioonid, nende seas ka OpenSZZ. Kuigi SZZ algoritmil on piiranguid ja mõned neist on juba ületatud, rakendab OpenSZZ selle algoritmi põhiversiooni. Erinevalt teistest SZZ teostustest on OpenSZZ pilvepõhine veebirakendus, ja ta toetab algoritmi mõlemat osa. Meie eesmärgiks on OpenSZZ edasi arendamine ning selle põhjal täiustatud SZZ versiooni väljatöötamine.
Tuvastasime OpenSZZ teostuses puuduseid, ja leidsime, et see annab mittekorrektseid

tulemusi. Antud lõputöös parandasime mõned OpenSZZ defektid. Lisaks arendasime OpenSZZ funktsionaalsust, tuginedes teistele SZZ algoritmi teostustele, ja lisasime muuhulgas võimaluse kasutada veahaldussüsteemi informatsiooni, et tuvastada vigu tekitavaid muudatusi. Lõpetuseks, lisasime võimaluse analüüsida repositooriumeid viisil, mis ei tugine veahaldussüsteemi informatsioonile.

Võtmesõnad:
SZZ algoritm, OpenSZZ, viga tekitav muudatus, viga parandav muudatus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

A bug in the context of software engineering is an error, flaw, or fault in a computer program or system that causes it to produce an incorrect or unexpected result or to behave in unintended ways.

Understanding how bugs are introduced may help to:

- Determine how much time does it take to identify and fix different types of bugs.
- Determine bug-prone change patterns.
- Identify bug-prone files and modules.
- Find circumstances when bugs are introduced more often (for example, days of the week, time of day).

Understanding how bugs are introduced and how they are fixed in software repositories may also help to avoid introducing bugs in the future. Machine learning-based approaches can be used to predict bug-prone commits and highlight commits that deserve meticulous code review. Yet, to predict bug-prone commits, a predictor needs to be trained on a dataset of existing bugs. Hence, bug-introducing commits need to be identified.

The problem of searching bug-introducing commits has been solved with the help of the SZZ algorithm, which can identify bug-fixing commits and find commits that introduced the bugs that were fixed. SZZ algorithm identifies bug-fixing commits by linking commits to issues tracked in an issue tracking system (ITS). The algorithm identifies bug-introducing commits using the annotation/blame feature of version control systems (VCS). For each line of a file this feature points to the commit which introduced the last changes in the line. The SZZ algorithm was called SZZ by the first letters of surnames of authors of the work where the algorithm was published for the first time - Śliwerski-Zimmermann-Zeller [1]. This initial version of the SZZ algorithm will be called the basic version of SZZ (B-SZZ).

Unfortunately, recent studies show that B-SZZ has limitations and may provide incorrect results. Results are considered incorrect if commits that do not introduce a bug are considered bug-introducing, and the real bug-introducing commits are missed.

Lenarduzzi *et al.* have recently published the OpenSZZ tool - a free open-source web-accessible implementation of B-SZZ. OpenSZZ is parallelizable and highly distributable for usage in large-scale production systems [2]. Authors of OpenSZZ propose their implementation of B-SZZ as a common ground for researchers to build and base their experiments and submit new versions of the SZZ algorithm on top of it.

1.1 Motivation

OpenSZZ is not the only free open-source implementation of the SZZ algorithm. Other implementations were also proposed. It would be good to combine the advantages of different SZZ implementation.

B-SZZ involves the usage of an issue tracking system. In most studies related to SZZ, only such repositories were analyzed where ITS was used, commits contain IDs of addressed issues in commit messages, and the ITS is publicly available. However, ITS is used not for all projects. And not for all projects where an ITS is used, the ITS is publicly available. Another approach to find bug-fixing commits with enough confidence and to locate bug-introducing commits accurately needs to be implemented and verified.

Why OpenSZZ was chosen to be improved among existing open-source SZZ implementations

OpenSZZ allows analyzing repositories with Jira[1] used as an issue tracker. OpenSZZ requires only 2 values to analyze repositories: URL of a GIT repository and URL of a Jira project used to track issues in this repository.

RA-SZZ, R-SZZ and other SZZ implementations proposed by Neto, Costa and Kulesza [3] do not provide a way to analyze projects by only specifying repository URL and issue tracker URL. Their implementations rely on the issues and bug-fixing commits data stored in a database. Although a database backup to restore all data related to the projects analyzed in their study is provided, analyzing new projects requires a preliminary database population with issues and corresponding bug-fixing commits. This part is not automated in the SZZ implementations by Neto, Costa and Kulesza, but it is automated in OpenSZZ.

SSZ-Unleashed [4] is another well-documented SZZ implementation, but it is not wrapped into a cloud-native realization with a web-accessible interface.

Lenarduzzi *et al.* provided a cloud-native version of OpenSZZ[2]. This way, their implementation is web-accessible and highly distributable. OpenSZZ-Cloud-Native allows analyzing multiple projects in parallel.

Although OpenSZZ is an implementation of B-SZZ and does not overcome some existing limitations of the SZZ algorithm, the implementation was chosen as a base to propose improvements of SZZ on top of it.

1.2 Goals

Lenarduzzi *et al.* evaluated OpenSZZ [2] and provided results of analyzing different repositories of Apache Software Foundation by OpenSZZ [5]. However, to base SZZ

[1] `https://www.atlassian.com/software/jira`

[2] `https://github.com/clowee/OpenSZZ-cloud-native`

algorithm improvements on top of OpenSZZ, OpenSZZ needs to be evaluated to confirm that results provided by the software are expected, and it works according to B-SZZ.

Since OpenSZZ is an implementation of B-SZZ, it has all the limitations of SZZ, which will be described later in section 2.2. The limitations need to be overcome to make OpenSZZ providing more accurate results.

OpenSZZ relies on Jira ITS to identify bug-fixing commits and more accurately identify bug-introducing commits. Therefore, OpenSZZ usage is limited to repositories with issues tracked with publicly available Jira. A way to analyze other repositories needs to be found.

This way, we formulated the next three goals for the work:

G1. Evaluate the correctness of OpenSZZ.

G2. Improve OpenSZZ - apply improvements proposed by other studies and overcome limitations.

G3. Improve OpenSZZ - make it possible to use it for repositories without ITS used.

2 Background

This chapter describes the SZZ algorithm and its limitations.

2.1 SZZ

The SZZ algorithm was presented by Śliwerski, Zimmermann, and Zeller [1]. And it consists of 2 parts:

1. Identifying bug-fixing commits. It is done with the help of an issue tracking system.
2. Locating bug-introducing commits. It is done with the help of the annotate/blame feature of a version control system.

2.1.1 Identifying bug-fixing changes

Commits in VCSs contain information on who, when, and what changed, but the purpose of changes can be identified solely with the commit message. A common practice among developers is to include an issue ID in the commit message when changes in this commit are related to the issue. More information about the purpose of changes can be obtained when commits are linked to issues in ITS.

Steps to identify bug-fixing commits:

1. Download the log of commits of a certain repository, download all issues from ITS.
2. From the list of commits, extract those that possibly contain an issue ID in their commit message.

   Issue IDs have different patterns in different ITS. Issue IDs in Jira always have the `<Jira project key>-<issue ID number>` pattern, which corresponds to the `[a-zA-Z]+-[0-9]+` regular expression.
3. Link filtered commits to corresponding issues based on the presence of issue IDs in their commit messages.
4. Perform syntactic and semantic analysis of each link to obtain corresponding confidence scores.
5. Filter links that reached a certain score of syntactic and semantic confidence and have a commit created before the issue was opened.

Syntactic analysis. Syntactic confidence score increases when the commit message matches the particular keywords: (regular expression) `fix(e[ds])?|bugs?|defects?|patch`.

Semantic analysis. Semantic analysis includes the next checks. Each check can increase the semantic confidence score by 1:

- The issue has been resolved as *Fixed* at least once.
- The commit message contains a part of the issue title
- The author of the commit has been assigned to the issue.
- One or more of the files affected by the commit have been attached to the issue.

Filtering commits using the semantic and syntactic confidence scores. SZZ proposes to consider commits to be bug-fixing, if they have

$$sem > 1 \cup (sem = 1 \cap syn > 0),$$

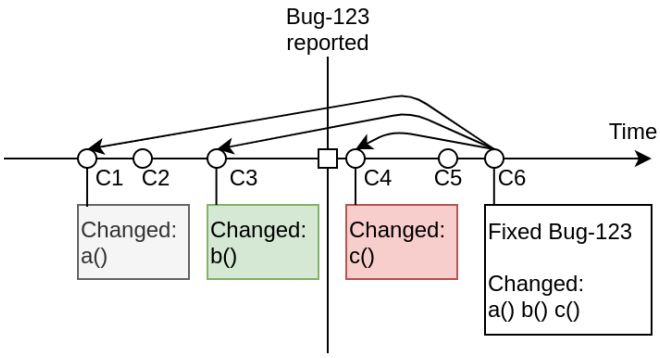
where *sem* - semantic confidence score, *syn* - syntactic confidence score.

2.1.2 Locating bug-introducing changes

The process of locating bug-introducing changes will be demonstrated with the example shown in Figure 1.

Steps to locate bug-introducing changes:

1. Find lines changed in bug-fixing commit using *git diff* command.
2. Call *git blame* command for the parent of bug-fixing commit - C5. It is the last state of the repository without the fix.
3. Ignore changes made after opening the issue. Let's call results of *git blame* for lines from step 1 suspects (hence, C1, C3 and C4 are suspects). They are the commits where the lines were edited the last time. For every line from step 1: if the suspect was committed after the Bug-123 was reported - ignore. Such changes could not contribute to the issue. (commit C4).
4. Among suspects left, select the latest one (C3).



C1-C6 are commits. C6 is a bug-fixing commit.

Figure 1. Locating bug-introducing changes

2.2 SZZ limitations

The SZZ algorithm needs to provide correct results to analyze repositories correctly and be used in just-in-time (JIT) defect prediction. However, recent studies show that SZZ results are not always correct, and SZZ has limitations [6][7]. The more limitations are overcome - the more correct results can be obtained.

This sections describes limitations of the basic version of SZZ.

Limitations related to identifying bug-fixing commits:

- Incomplete mapping. A bug report cannot be linked to a commit that addressed the bug. A commit is related to the issue but does not contain the issue ID in the commit message [8].
- Inaccurate mapping. The fixing commit has been linked to a wrong bug report, and they do not correspond to each other [9].

Changes that do not affect the software's behaviour and should be ignored.

Some changes may be considered as bug-fixing changes, but they are not related to the bug fix. Some changes may be considered candidates for bug-introducing changes, but they are not related to the bug. Such changes are:

- Formatting and changes in comments [10]. An example of such changes is demonstrated in Figure 2.
- Refactoring. Williams and Spacco found that 6.5% of bug-introducing lines of the projects they analyzed were refactoring changes. 19.9% of lines modified

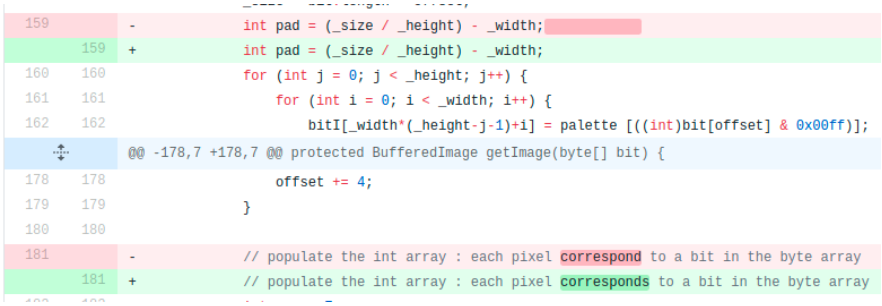


Figure 2. Formatting and changes in comments.

during bug fixes were refactoring changes [10]. Results are noisy when refactoring changes are not ignored.

An example of refactoring is demonstrated in Figure 3.

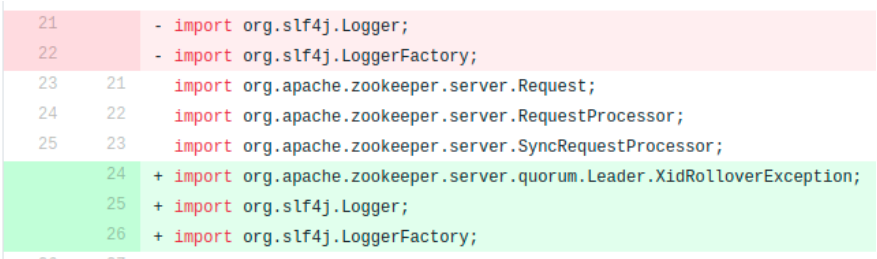


Figure 3. Refactoring changes.

- Semantically equivalent changes [7]. Replacing old language constructions with equivalent new ones does not affect software's behaviour. As with formatting changes, different programming languages need to be taken into account. An example of semantically equivalent changes is demonstrated in Figure 4.

Situations that may lead to incorrect results:

- Bug was fixed by editing code not edited before by bug-introducing commit. Example: `getMainArguments` function of `LauncherMapper.java` file may return `null` as an element of a returned list (introduced with 8367900b[3]). Function `printArgs` of `LauncherMapper.java` file (added with 3276633f[4]) throws `Null pointer exception` once method `toLowerCase()` is called on an argument with

[3] `https://github.com/apache/oozie/commit/8367900b`
[4] `https://github.com/apache/oozie/commit/3276633f`

```
247  247          result[0] = Integer.MAX_VALUE;
248       -       zknew.sync("/", new AsyncCallback.VoidCallback() {
249       -           public void processResult(int rc, String path, Object ctx) {
250       -               synchronized (result) {
251       -                   result[0] = rc;
252       -                   result.notify();
253       -               }
     248  +       zknew.sync("/", (rc, path, ctx) -> {
     249  +           synchronized (result) {
     250  +               result[0] = rc;
     251  +               result.notify();
254  252          }
```

Figure 4. Semantically equivalent changes.

the value `null`. Instead of fixing `printArgs` function, the bug was fixed with changes in `getMainArguments` function (c7fa12bb[5]). The commit which introduced `getMainArguments` function and is considered as bug-introducing commit according to SZZ did not introduce the bug by itself and was correct on the moment of its creation.

- When repository commits address issues from multiple issue tracking systems (ITS). In this case, the number of bug-fixing commits can be limited if they are searched by a single key.
- Extrinsic bugs. Extrinsic bugs - bugs that were introduced by changes not registered in the VCS. They may appear due to changes in external dependencies, environment changes, or changes in requirements. Such bugs do not have a bug-introducing commit. The code was correct at the moment of writing it [11].
- Reverted changes. Such blackout changes create a problem because the reverted code will be considered as introduced in the reverting commit [12].

Suspicious cases that need to be analyzed meticulously:

- Bug-fixing commits with a big number of changed files. Zimmermann, Pan and Whitehead consider the number of changed files big when it is five times bigger than the median number of files changed in commits [6].
- Changes that cause a high number of bug-fixing changes.
- Changes that have a large time span between them but were edited in a single bug-fix commit.

[5] `https://github.com/apache/oozie/commit/c7fa12bb`

If Jira is used as ITS, the "Affects Version/s" issue property can be used to improve the search of bug-introducing commits. Knowing the earliest software's version where the issue occurs can help determine the earliest date when bug-introducing changes can be found. Costa *et al.* found that only 4% of analyzed issues in their study had the value set [7].

Bugs can be fixed by adding new lines of code and without deleting/modifying existing code. SZZ cannot find bug-introducing commits in the cases. Identifying false negatives (i.e., bug-introducing changes that are not flagged as such by SZZ) remains an open challenge [7].

Wen *et al.* described the limitations of SZZ and proposed another approach to find bug-introducing commits [13]. They search bug-introducing commits relying on tests. A commit that causes a test fail is considered a bug-introducing commit. Bug-introducing changes are the code lines modified in a bug-fixing commit. However, this approach may be applied to an even much smaller number of repositories since it requires the entire code base to be covered with tests.

2.3 SZZ variants

Previous sections 2.1 and 2.2 described B-SZZ. However, the SZZ algorithm has improved over time. New versions that address some of its limitations were implemented. The evolution of SZZ is shown in Figure 5. A description of the different versions and their implementations can be found below.

Basic SZZ (B-SZZ) is the initial version of the SZZ algorithm presented by Śliwerski *et al.* [1]. OpenSZZ is an implementation of B-SZZ.

Annotation Graph SZZ (AG-SZZ) is a version that implies the usage of an annotation-graph to find potential bug-introducing changes. The annotation graph is used to represent the evolution of each line of code within source files. The idea is not to ignore the lines modified after opening the issue but trace them further to reach commits where the lines were edited before opening the issue. The improvement was proposed by Kim *et al.* [14] and implemented by Costa *et al.*[7].

SZZ Unleashed is another implementation of AG-SZZ [4].

Meta-change Aware SZZ (MA-SZZ) is a version that implies ignoring meta-changes - changes that do not affect software's behaviour. MA-SZZ implemented by Costa *et al.* on top of AG-SZZ ignores changes in comment lines of Java files and changes in end-of-line characters [7].

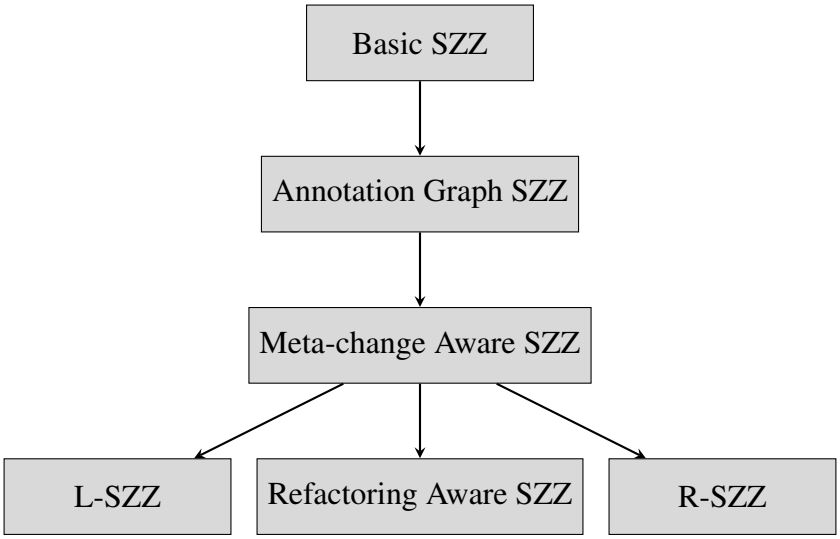


Figure 5. SZZ evolution.

Refactoring Aware SZZ (RA-SZZ) is a version that implies ignoring refactoring changes. Neto, Costa and Kuleza detect the refactoring changes with the help of the RefDiff tool [15] in their RA-SZZ implementation [3]. The next year after presenting RA-SZZ they revisited the implementation and proposed to use RefactoringMiner [16] instead of RefDiff [17]. RA-SZZ can be applied only to Java repositories because it mines code changes using the DiffJ tool to ignore changes in comments, formatting and whitespace changes [18], and only Java files can be compared with DiffJ.

R-SZZ is a version that implies considering the most recent potential bug-introducing change as bug-introducing for each file changed in a bug-fixing commit. The improvement was proposed by Davies *et al.* [19] and implemented by Costa *et al.* [7] on top of MA-SZZ.

L-SZZ is a version that implies considering the largest potential bug-introducing change instead of the most recent one as bug-introducing. The improvement was proposed by Davies *et al.* [19] and implemented by Costa *et al.* [7] on top of MA-SZZ.

3 Method

This chapter describes how the goals G1 to G3 (defined in section 1.2) can be achieved.

3.1 Evaluation of OpenSZZ correctness

This section describes how OpenSZZ correctness will be evaluated. If issues with OpenSZZ implementation are found (results provided by OpenSZZ are unexpected), they will be resolved.

OpenSZZ can be evaluated by comparing its results for a certain project with correct results for the project. There are two ways to obtain the correct results to be used as a benchmark:

- Check results ourselves (as if we are domain experts) - manual verification.
- Compare with results verified by domain experts from previous studies related to SZZ.

The verification is needed to ensure that OpenSZZ provides expected results.

3.1.1 Manual verification

To verify the results of OpenSZZ manually, we need to know which commit should be provided by *git blame* for certain changes.

It can be checked in GitHub this way:

1. Open a bug-fixing commit (BFC).
2. Open the parent commit of the BFC.
3. Open *Browse files* mode and open the target file. This way, the last state of the file before the BFC can be viewed.
4. Open *Blame* mode for the file.
5. Find commits blamed to be responsible for the latest changes in the lines changed in the BFC. Choose a commit that is the most recent and was created before the referenced in BFC issue was opened.

3.1.2 Verification on datasets used in previous studies

Wen *et al.* explored seven Apache projects [13]. Their benchmark datasets of bug-fixing commits and the associated bug-inducing commits are publicly available[6].

To construct the benchmark datasets, Wen *et al.* did not rely on their software development knowledge. They constructed the datasets based on the information provided by domain experts in bug reports in the next three steps:

1. Select bug reports that satisfy one of the following conditions:
   (a) The report has a specific attribute that indicates that other issues introduced this bug. In Jira, the attribute is "is broken by" under the "Issue Links" category.
   (b) The report description or comments match the next regular expression:

```
(started with|caused by|introduced by|commit).*?(r*(\w){6,41}|PROJECT-(\d){3,7})
```

2. Analyze the reports manually and discard such reports where matched texts do not provide information on commits that introduced the bug or issues addressed by the bug-introducing commit.

3. Link bug reports to the commits that contain the report ID in the commit message. Discard bug reports if the bug-fixing commit was not found this way. Discard commits with the issue ID in the commit message but are not related to the bug fix.

The benchmark datasets can be used to determine whether bug-introducing commits found by OpenSZZ for issues present in the them are correct.

Note that benchmark datasets by Wen *et al.* are not complete datasets of issues in selected projects. The number of issues for each dataset is limited. Bug-introducing commits are provided only for some part of issues existing in projects. Based on this datasets limitation, the next aspects cannot be analyzed:

- How many issues present in the project were not found by OpenSZZ.
- How many false issues were found by OpenSZZ.

Oozie[7] project can be chosen for analysis because it has the smallest number of commits in its repository with relatively the same number of issues in the benchmark dataset. A smaller number of commits in the repository allows us to analyze it with OpenSZZ faster.

[6] https://github.com/justinwm/InduceBenchmark
[7] https://github.com/apache/oozie

Oozie project has 1496 issues with the type *Bug* and resolution *Fixed*. But provided benchmark dataset has information only about 44 such issues. OpenSZZ-original provides results for all issues in the issue tracker, not only for issues with the type *Bug*. Thus results for issues with types *task*, *subtask*, *new feature*, *improvement* could be included. Since we are interested only in results for bug issues - issues with other types are ignored.

The structure of the benchmark dataset is shown in Table 1.

Table 1. Benchmark dataset structure.

Column name	**Column value**
BugID	ID of an issue addressed by bug-fixing commit(s)
BugFixingCommit	Set of bug-fixing commits separated by a comma, where each commit is presented by the first eight characters of its hash
BugInducingCommit	Set of bug-introducing commits separated by a comma, where each commit is presented by the first eight characters of its hash

Each row has *BugID* as a key - it is always a single unique value. Both *BugFixingCommit* and *BugInducingCommit* may contain multiple commits as a value in each row, but multiple bug-fixing commits for the same issue are rather exceptional cases.

The Structure of an OpenSZZ result dataset is shown in Table 2.

Table 2. OpenSZZ result dataset structure.

Column name	**Column value**
bugFixingId	A full bug-fixing commit hash (40 characters)
bugFixingTs	Creation date of the bug-fixing commit in `yyyy-MM-ddTHH:mm:ss` format
bugFixingFileChanged	Path to a file changed by the bug-introducing commit and then changed by the bug-fixing commit
bugIntroducingId	A full bug-introducing commit hash (40 characters)
bugIntroducingTs	Creation date of the bug-introducing commit in `yyyy-MM-ddTHH:mm:ss` format
issueId	ID of an issue addressed by the bug-fixing commit

OpenSZZ result dataset does not have a column that contains unique values and could be treated as a key, like *BugID* column in a benchmark dataset. Only the combination of *bugFixingId*, *bugFixingfileChanged*, *bugIntroducingId*, and *issueId* values is unique in the dataset. Values *bugFixingTs* and *bugIntroducingTs* are redundant and only provide the timestamp information for corresponding commits. OpenSZZ searches a bug-introducing

commit for each file changed in each bug-fixing commit. Therefore, it is quite possible that for a bug fixing commit with multiple changed files, OpenSZZ returns multiple bug-introducing commits (a maximum of one bug-introducing commit per changed file).

An OpenSZZ dataset and a benchmark dataset need to have an equal structure to be compared.

The correctness of the OpenSZZ result is evaluated by comparison BICs for certain BFCs in the result to BICs of the same BFCs in the benchmark dataset. The common dataset structure to compare an OpenSZZ result dataset and a benchmark dataset is shown in Table 3.

Table 3. The common dataset structure to compare an OpenSZZ result dataset and a benchmark dataset.

Column name	**Column value**
BugFixingCommit	A commit presented by the first eight characters of its hash
BugInducingCommit	Set of bug-introducing commits separated by a comma, where each commit is presented by the first eight characters of its hash

To be compared with OpenSZZ result dataset, *BugFixingCommit* and *BugInducingCommit* columns are extracted, and rows with multiple bug-fixing commits are removed.

OpenSZZ result dataset is transformed to the common dataset structure in the next steps:

1. Reduce commit hashes from 40 to the length of commit hashes in the benchmark dataset.
2. Extract a list of unique *bugFixingId*.
3. For each *bugFixingId* combine a list of unique *bugIntroducingId* values.

Now, when both the OpenSZZ result dataset and a benchmark dataset have the same structure, they can be compared.

The correctness of the OpenSZZ result is evaluated by comparison bug-introducing commits. A set of bug-introducing commits from OpenSZZ result for a single bug-fixing commit is referred to as a predicted result. A set of bug-introducing commits from a benchmark dataset for a single bug-fixing commit is referred to as an actual result. This way, the OpenSZZ result dataset has a predicted result for each bug-fixing commit, a benchmark dataset has an actual result for each bug-fixing commit. Since both predicted and actual results can contain multiple values, presenting a result of comparison with a boolean value does not suit well in this case. Sensitivity and precision measures are used to present the result of the comparison predicted result to the actual result.

Sensitivity shows the rate of correctly predicted bug-introducing commits among all actual bug-introducing commits. Precision shows the rate of correctly predicted bug-introducing commits among all predicted bug-introducing commits.

To calculate sensitivity, true positives and false negatives need to be known. To calculate precision, true positives and false positives need to be known.

True positives - bug-introducing commits from a predicted result that are present in the actual result.

False positives - bug-introducing commits from a predicted result that are not present in the actual result.

False negatives - bug-introducing commits from actual result not present in the predicted result.

Sensitivity is calculated as

$$\begin{aligned} Sensitivity &= \frac{TP}{TP + FN}, \\ \text{where } TP &= \text{number of true positives}, \\ FN &= \text{number of false negatives} \end{aligned}$$

Precision is calculated as

$$\begin{aligned} Precision &= \frac{TP}{TP + FP}, \\ \text{where } TP &= \text{number of true positives}, \\ FP &= \text{number of false positives}, \end{aligned}$$

The sensitivity and precision of bug-introducing commits are calculated for each bug-fixing commit and then aggregated to average sensitivity and average precision.

The number of bug-fixing commits from the benchmark dataset with bug-introducing commits found by OpenSZZ, their average sensitivity and average precision are the characteristics used to compare different versions of OpenSZZ. The higher these characteristics are, the more accurate result can be obtained by the particular OpenSZZ version.

Example of calculating OpenSZZ metrics

Let's have a benchmark dataset containing three issues. The dataset is presented in Table 4.

Let's have an OpenSZZ result dataset containing four bug-fixing commits. The dataset after transformation for comparison is presented in Table 5.

Table 4. Example of a benchmark dataset after transformation for a comparison.

BugFixingCommit	**BugInducingCommit**
0a6f83e6	81ce22b6,87040a1f
db11ab35	cc4b4398
aebf775a	0f086d41

Table 5. Example of an OpenSZZ result dataset after transformation for a comparison.

BugFixingCommit	**BugInducingCommit**
0a6f83e6	81ce22b6
db11ab35	9501311a,f82c1240
c7fa12bb	8367900b
d361ee4d	c67b29f6

OpenSZZ result dataset contains two bug-fixing commits present in the benchmark dataset - `0a6f83e6` and `db11ab35`. Table 6 shows how sensitivity and precision are calculated for the bug-fixing commits.

Table 6. Calculating sensitivity and precision for OpenSZZ results.

BFC	0a6f83e6	db11ab35
Predicted	81ce22b6	9501311a,f82c1240
Actual	81ce22b6,87040a1f	cc4b4398
True positives	81ce22b6 (1)	(0)
False positives	(0)	9501311a,f82c1240 (2)
False negatives	87040a1f (1)	cc4b4398 (1)
Sensitivity	$1/(1+1) = 0.5$	$0/(0+1) = 0$
Precision	$1/(1+0) = 1$	$0/(0+2) = 0$

Having sensitivity and precision calculated for each bug-fixing commit present in both the OpenSZZ result dataset and benchmark dataset, we can calculate average sensitivity as a sum of all sensitivity values divided by the number of them and average precision as a sum of all precision values divided by the number of them:

Average sensitivity: $(0.5 + 0)/2 = 0.25$.

Average precision: $(1 + 0)/2 = 0.5$.

The number of bug-fixing commits in the OpenSZZ result dataset with found bug-introducing commits: 4.

The number of bug-fixing commits from the benchmark dataset with bug-introducing commits found by OpenSZZ: $2/3(0.667)$.

Calculation of the metrics is implemented in OpenSZZ-evaluation tool[8], which takes an OpenSZZ result dataset and a benchmark dataset as input.

3.2 Improvement of OpenSZZ with Issue Tracker

This section describes how OpenSZZ will be improved by addressing the limitation of B-SZZ - "Bug-fixing commit may contain changes not related to fixing a bug". The limitation will be addressed in the next steps:

1. Ignoring whitespace changes.
2. Ignoring changes in comment lines.
3. Ignoring formatting and comment changes in Java files.
4. Ignoring refactoring changes.
5. Searching for bug-introducing commits based on information provided by domain experts in Jira issues

This section also describes how additional information about bugs from Jira issues can be used to find bug-introducing commits.

3.2.1 Ignoring whitespace changes

This section describes how whitespace changes and blank line changes can be ignored.

Whitespace changes.
Whitespace changes can be categorized as such that are related to fixing a bug and such that are unrelated to fixing a bug:

- **Whitespace changes related to fixing a bug**. An example of such changes are changes in `SqoopActionExecutor.java` file of commit `a98c7f89`[9] from Oozie repository. The Result of the default *git diff* command is shown in Figure 6. The result of *git diff -w* (ignoring all whitespace changes) is shown in Figure 7.

  It is visible in Figure 7, that bug in this file was fixed with only line additions. In some cases code addition fixes are represented by wrapping existing code blocks into new code lines (`if-else`, `try-catch` blocks). These changes lead to indentation changes of existing code blocks (they are called leading whitespace changes). The next assumption was made: old code lines wrapped into new code

[8] `https://github.com/VladyslavBondarenko/OpenSZZ-evaluation/tree/v2`
[9] `https://github.com/apache/oozie/commit/a98c7f89`

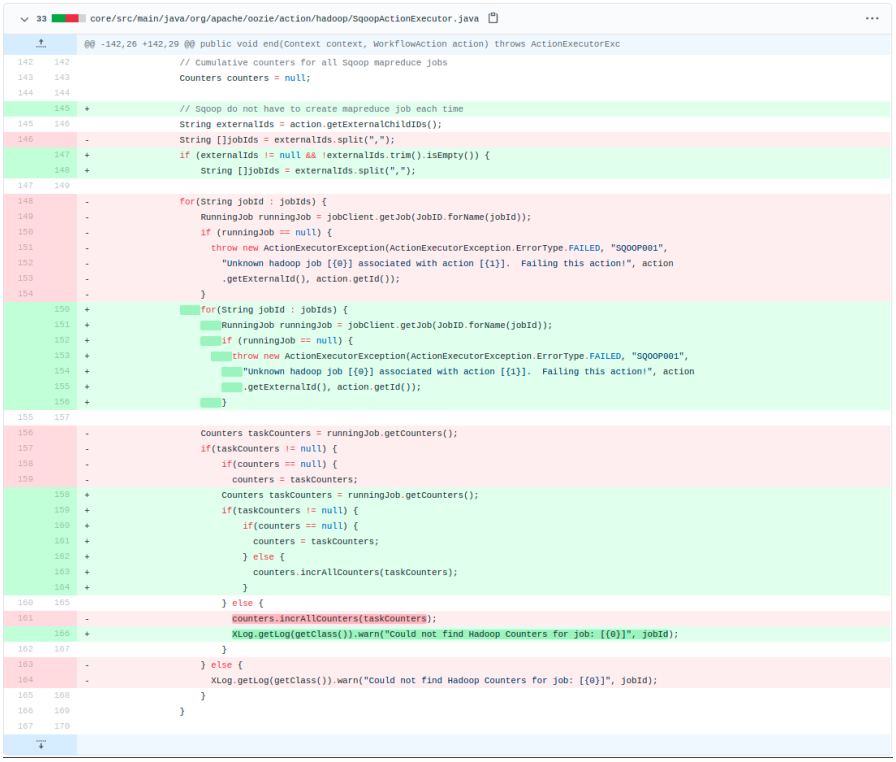


Figure 6. Result of *git diff* command.

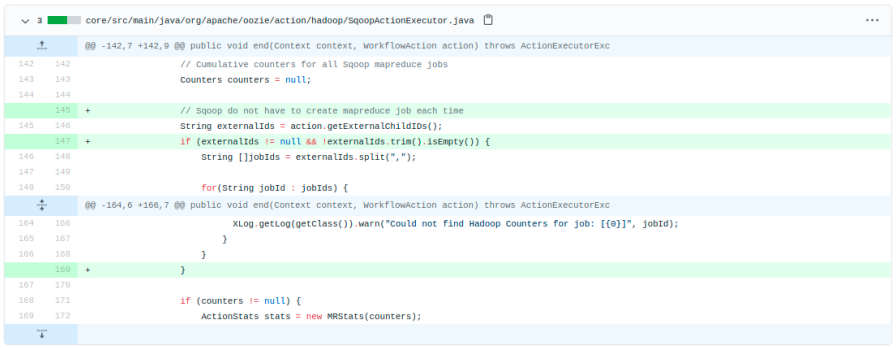


Figure 7. Result of *git diff -w* command.

lines in a bug-fixing commit are bug-introducing code lines. Hence, commits that inserted the old code lines are possible bug-introducing commits. With this assumption, such indentation changes will not be ignored. It will allow finding a bug-introducing commit when a bug was fixed with only code additions, which caused indentation changes in old code.

- **Whitespace changes unrelated to fixing a bug**. An example of such changes are changes in `JavaActionExecutor.java` file of commit `a9c26541`[10] from Oozie repository. Figure 8 shows part of the commit with whitespace changes unrelated to fixing a bug.

Figure 8. Example of whitespace changes unrelated to fixing a bug

It is visible in Figure 8, that changes in lines 248, 249, 270, 271, and 417 do not influence software's behaviour. The changes should be ignored.

Solution. Whitespace changes can be ignored with help of setDiffComparator function of DiffFormatter class from JGit library - a Java library for work with git-repositories[11].

However, JGit has limitations. The library provides only 5 options for *git diff* command[12]:

1. No special treatment.
2. Ignore all whitespace.
3. Ignore whitespace occurring between non-whitespace characters.
4. Ignore leading whitespace.
5. Ignore trailing whitespace.

[10] `https://github.com/apache/oozie/commit/a9c26541`

[11] `http://archive.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/diff/DiffFormatter.html#setDiffComparator(org.eclipse.jgit.diff.RawTextComparator)`

[12] `http://archive.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/diff/RawTextComparator.html`

Option 2 includes 3, 4, and 5.

It would be helpful to use 3 and 5, but not 4, because leading whitespace changes may help identify bug-introducing commits for bugs fixed with wrapping existing code into new code lines. But only a single option may be chosen. Therefore, only trailing whitespace changes will be ignored as changes that definitely do not influence software's behaviour and changes that happen more often than whitespace changes occurring between non-whitespace characters.

Blank line changes.

An example of a blank line changes is the commit 19f87e2 from Oozie repository[13]. The changes of this commit are shown in Figure 9.

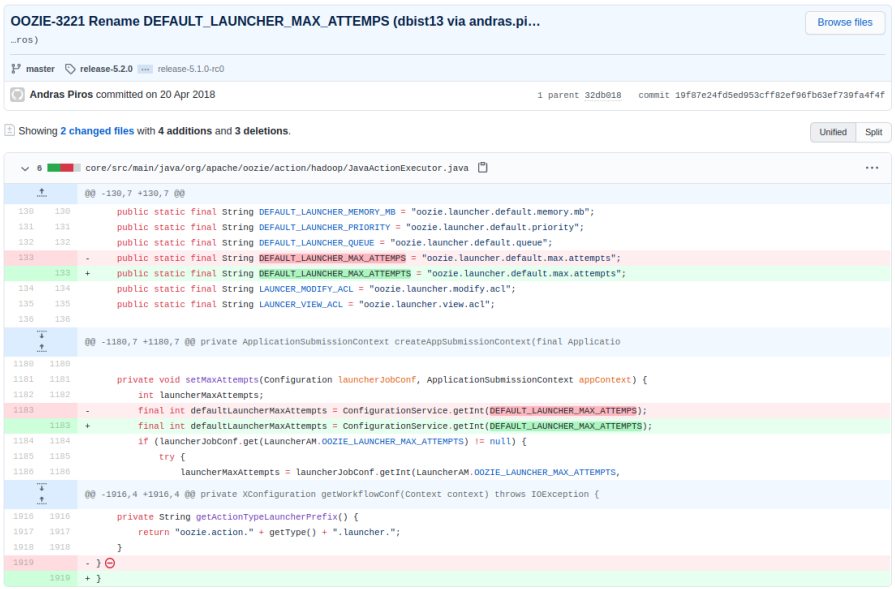


Figure 9. BFC with a blank line change.

Issue title is "Rename DEFAULT_LAUNCHER_MAX_ATTEMPS". Changes in lines 133 and 1183 address the typo. But the change in line 1991 only appends a new line to the end of the file and is not related to fixing the bug. Among the three changed lines, line 1991 was changed the latest before the BFC. Hence, the commit that introduced a previous change to the line (removed a new line at the end of the file) was chosen as BIC by OpenSZZ-corrected. Change in line 1991 should be ignored.

Solution. JGit doesn't have an exact analog of *--ignore-blank-lines* parameter for *git diff* command. But changes in blank lines can be ignored in another way. All deleted

[13] `https://github.com/apache/oozie/commit/19f87e2`

blank lines in *git diff* are represented as lines with the only symbol - "-". This way, lines with a length equal to one can be ignored.

Conclusion

OpenSZZ has a limitation that bug-introducing commits can be searched for changes not related to fixing bugs. Some of such changes are whitespace changes and changes in blank lines.

The limitation will be addressed in a new version of OpenSZZ - modified OpenSZZ-corrected. The new OpenSZZ version that ignores trailing whitespace changes and blank line changes in commit changes on the step of searching bug-introducing commits will be called **OpenSZZ-IWS**.

3.2.2 Ignoring changes in comment lines

Changes in comments do not affect software's behaviour. Therefore, changes in comments of bug-fixing commits should be ignored on the step of searching for bug-introducing commits.

There are two types of comments:

- Inline comment - any text between a special character sequence (inline comment beginning) and the end of the line.
- Block comment - any text between a special character sequence (block comment beginning) and another special characters sequence (block comment end).

Different programming languages have different character sequences to define comments. Table 7 shows what character sequences are recognised as comments in different types of files. The list of file extension presented in the table covers file extensions of the first 24 most popular programming languages on GitHub[14].

Listing 1 shows possible ways of writing comments in Java. This example is also applied to other programming languages as long as the comments-related character sequences correspond to the programming languages according to Table 7.

An OpenSZZ version that ignores changes in comment lines will be called **OpenSZZ-IC**. This new version of OpenSZZ is a modified OpenSZZ-IWS.

Although it is possible to identify all types of comments, determining whether a change is made in a comment or in code is impossible with the current SZZ implementation. OpenSZZ uses JGit Java library to interact with git-repositories. The library allows getting results of *git diff* command that shows new and old versions of each changed line. However, the command does not show an exact change location in the string.

[14] `https://madnight.github.io/githut/#/pull_requests/2021/1`

Table 7. Regular expressions to detect comment lines in different file types.

File extension	Start inline comment	Start block comment	Finish block comment
c, java, kt, cs, cpp, h, go, js, ts, swift, m, mm, r, scala, sc, dart, dm, groovy, gvy, gy, gsh, coffee	\\	/*	*/
php	\\ | #		
css, scss, sass, less			
sql	--		
lua		--[[	--]]
rs	\\		
sh, bash, zsh	#	† COMMENT	† «COMMENT
py, ex, exs		""" | '''	""" | '''
rb		=begin	=end
coffee		###	###
pl		† =	† =cut
html, htm, xml		<!--	-->

† means that the sequence can be only at the line beginning.

```
// inline-comment
/* single-line block comment */

/*
 * block comment
 */

int a = 0; // trailing inline-comment
int b = 0; /* trailing single-line block comment */
int /* block comment within code */ c = 0;
/* leading single-line block comment */ int d = 0;
```

Listing 1. Ways of writing comments in Java code

Exact differences in strings can be obtained using *git diff* command with the *–word-diff* flag, but JGit does not support this command. Therefore, OpenSZZ-IC can ignore changes in lines that have only comments and cannot ignore changes in lines that have a code with comments despite changes were made in comments.

Table 8 shows what types of comments are ignored by OpenSZZ-IC. The types of

comments are demonstrated in Listing 1.

Table 8. Changes in comments ignored by OpenSZZ-IC.

Comment type	**Ignored by OpenSZZ-IC**
Inline-comment	✓
Single-line block comment	✓
Block comment	✓
Trailing inline-comment	
Trailing single-line block comment	
Block comment within code	
Single-line block comment followed by code	

With comment lines changes ignored, a bug-fixing commit can have a smaller number of changed lines. This may lead to decreasing the number of candidates for bug-introducing commits (by excluding the knowingly non-bug-introducing candidates). This way, chances to identify correct bug-introducing commits increase.

3.2.3 Ignoring formatting and comment changes in Java files

OpenSZZ-IC ignores trailing whitespace changes but does not ignore whitespace changes within code. Also, OpenSZZ-IC ignores not all comment changes but only comment lines without a code. Therefore, to find bug-introducing commits, OpenSZZ-IC traces back also such lines that do not influence software's behaviour.

Neto, Costa, and Kulesza use the DiffJ tool in their RA-SZZ* implementation to mine changes in Java code [17]. DiffJ is a Java syntax-aware diff tool that compares Java files based on their code, without regard to formatting, organization, comments, or whitespaces [18].

DiffJ is a command-line tool and is not intended to be used as a module of another project. The result of file comparison is encapsulated and is accessible only from within the package. Therefore, a minor change was required to allow access to the comparison results from an external package (OpenSZZ in the case)[15].

An OpenSZZ version that is built on top of OpenSZZ-Corrected and uses DiffJ to ignore non-executable changes in Java file will be called **OpenSZZ-DJ**.

3.2.4 Ignoring refactoring changes in Java files

Every bug-fixing commit should be checked on the presence of refactoring changes. Such changes should be ignored and not be used for searching bug-introducing commits.

[15] `https://github.com/VladyslavBondarenko/diffj/commit/4f4ee69`

Neto, Costa, and Kulesza analyzed Java repositories and used RefDiff[16] and Refactoring Miner[17] in their RA-SZZ* implementation. They concluded that Refactoring Miner is more effective [17]. Therefore, Refactoring Miner will be integrated into OpenSZZ.

The idea behind using a tool to detect refactoring changes is to ignore changes that do not influence software's behaviour. Refactoring Miner v1.0 used in RA-SZZ* can detect 15 refactoring types. The latest Refactoring Miner version at the moment is v2.1, and it supports already 62 refactoring types. However, only some of the refactoring changes detected by Refactoring Miner do not influence software's behaviour, and only they need to be taken into account.

OpenSZZ-IC modified in the way to ignore refactoring changes will be called **OpenSZZ-RA** (refactoring aware). The list of refactoring changes that do not influence software's behaviour and are ignored by OpenSZZ-RA:

1. *Extract Variable*
2. *Inline Variable*
3. *Replace Variable with Attribute*
4. *Rename Variable* (if a name change is the only change in variable definition)
5. *Rename Attribute* (a visibility change and a change in the presence of the Final keyword are allowed)
6. *Extract Attribute*
7. *Move and Rename Attribute* (a visibility change and a change in the presence of the Final keyword are allowed)
8. *Move Attribute* (a visibility change and a change in the presence of the *Final* keyword are allowed)
9. *Pull Up Attribute* (a visibility change and a change in the presence of the *Final* keyword are allowed)
10. *Push Down Attribute* (a visibility change and a change in the presence of the *Final* keyword are allowed)
11. *Rename Parameter* (if a name change is the only change in parameter definition and parameter position is the same)
12. *Reorder Parameter*

[16] `https://github.com/aserg-ufmg/RefDiff`
[17] `https://github.com/tsantalis/RefactoringMiner`

13. *Extract Method*

14. *Rename Method* (if a name change is the only change in method signature)

15. *Rename Class* (if a name change is the only change in class signature)

It is possible to find references to variables or attributes with Refactoring Miner. This way, all lines with referenced renamed variables or attributes are considered as refactoring changes as well.

For *Rename Class* refactoring, lines with signatures of constructors of the class are considered to have refactoring changes as well because constructors were renamed due to class renaming.

Limitations of Refactoring Miner:

- The main limitation is that Refactoring Miner can be applied to Java projects only.
- Method renaming causes changes in all locations where the method was called. All calls of the renamed method within the same class can be found with Refactoring Miner and ignored. However, all calls of the renamed method in classes other than the class where the method was defined are not interpreted as refactoring changes.
- Class renaming causes changes in import statements where the class was imported and in locations where static methods of the class are used. Such changes are not interpreted as refactoring changes. Class renaming causes the renaming of constructors of the class. Renaming constructors cause changes in all locations where the constructors were called (instances of the class were created). And such changes are not interpreted as refactoring changes.
- *Move Class*, *Extract Class*, *Extract Subclass*, *Move Method*, *Inline Method*, *Pull Up Method*, *Push Down Method*, *Extract and Move Method*, *Move and Inline Method* refactoring changes were skipped because Refactoring Miner does not imply exact match of classes or methods before and after commit. The class or method still can have other non-refactoring changes, and they should not be ignored.

3.2.5 Searching for bug-introducing commits based on information provided by domain experts in Jira issues

Although Jira reports are called issues, they are not only bug reports. Issues can have different types, which are used to distinguish different types of work. Default issue types for software projects in Jira are:

- Epic
- Bug
- Story
- Task
- Subtask

Issues with the type *Bug* will be called bug-reporting issues.

Using links to other issues to find bug-introducing commits

Issues of any type can be linked to other issues of any type by different link types. Such linking allows creating an association between issues on either the same or different Jira servers. For instance, an issue may duplicate another one, or its resolution may depend on another issue. Jira has four default types of links:

- Relates to / relates to
- Duplicates / is duplicated by
- Blocks / is blocked by
- Clones / is cloned by

The list of default link types can be extended with custom types.

One of such custom link types used for projects of Apache Software Foundation [18] is "breaks / is broken by" link type. When issue A is linked to the issue B by the "is broken by" link, it means that changes in the code which addresses the issue B introduced a bug reported by issue A. Issue B, in turn, is linked to the issue A by the "breaks" link. An example of an issue that has an "is broken by" link is OOZIE-2788[19].

The presence of the links "is broken by" in a bug-reporting issue may help to find bug-introducing commits for a bug-fixing commit addressing the bug-reporting issue in the next steps:

1. Collect IDs of the issues linked by "is broken by" links.
2. Search for commits that contain one of the issue IDs in their commit messages.
3. If such commits are not found, they are considered as bug-introducing commits. Otherwise, search for bug-introducing commits continues with the default approach - by using the blame feature of Git.

[18] `https://issues.apache.org/jira/secure/BrowseProjects.jspa`
[19] `https://issues.apache.org/jira/browse/OOZIE-2788`

An OpenSZZ version that uses issue links to find bug-introducing commits on top of OpenSZZ-RA will be called **OpenSZZ-UIL**.

Limitations of using links between Jira issues to find bug-introducing commits:

- "Breaks / is broken by" link type is a custom link type and is used possibly in a limited number of Jira projects.
- Not all bug-reporting issues have "is broken by" links to other issues.
- ID of the issue linked by "is broken by" link may be absent in messages of commits addressing the issue.

Analyze issue description and comments to find bug-introducing commits

References to other Jira issues equivalent to "is broken by" can be found in the issue description and comments by applying a specific regular expression to them. While this approach may have higher recall as this is a more common way for domain experts to link issues to each other, it may have lower precision since referenced issues may contain noise. Wen *et al.* exploited this approach to create benchmark datasets mentioned in section 3.1.2, but they also examined results manually to reduce noise [13].

Jira API provides the issue description and comments as HTML. To facilitate applying a regular expression to the description or comments, their values provided by Jira API need to be parsed to exclude HTML tags and extract text only. This can be done using Jsoup Java library[20].

An example of a Jira issue with the "introduced in" link in the issue description is OOZIE-3315[21]. Description of the issue contains the next phrase: "*The first element is missing in the list (but not the separator). This is caused by an off-by-one error introduced in OOZIE-2942*". This is the case when analyzing issue description may help to identify the bug-introducing commit because OOZIE-3315 does not have OOZIE-2942 linked as "is broken by", but it has the issue linked in the description.

An issue, implementation of which introduced a bug, will be called a bug-introducing issue. A regular expression to match such bug-introducing issue consists of two parts:

1. Match some target keywords followed by an issue ID.
2. Match an issue ID followed by some target keywords.

Issue ID is matched when any of the two parts is true. The keywords used in the part 1 will be called `matchBefore`. The keywords used in the part 2 will be called `matchAfter`.

A regular expression to match a bug-introducing issue (Issue-RE) is the following:

[20] `https://jsoup.org/`

[21] `https://issues.apache.org/jira/browse/OOZIE-3315`

```
(?<=matchBefore)issueID|issueID(?=matchAfter)
```

Where `issueID` is:

```
jiraProjectKey[ ]*-[ ]*[0-9]+
```

`matchBefore` is:

```
 (((introduc(ed|ing)|started|broken) ((this|the) (bug|issue|error) )?
(in|by|with)|caused by|due to|after|before|because( of)?|since) )
```

`matchAfter` is:

```
((introduced|caused)|[^.,:;]* cause))
```

Commits are identified by their hash. A regular expression to match the hash of a bug-introducing commit consists of two parts:

1. Matching some target keywords followed by a commit hash.
2. Matching a commit hash followed by some target keywords.

Commit hash is matched when any of the two parts is true. A regular expression to match a bug-introducing commit (Commit-RE) is the following:

```
(?<=matchBefore)commitHash|commitHash(?=matchAfter)
```

Where `matchBefore` and `matchAfter` are the same used in Issue-RE. And `commitHash` is:

```
(\b|(?<=(\br)))[0-9a-f]{5,40}\b
```

Applying Issue-RE to bug-introducing issue description and comments provides a list of issue IDs. Commits that have the issue IDs in their commit messages are considered as bug-introducing commits if they have edited files other than `.txt` and `.md`. Applying Commit-RE to bug-introducing issue description and comments provides a list of commit hashes. If commits with such hashes exist, they are considered bug-introducing.

Limitations of using Jira issue description and comments to find bug-introducing commits:

- Domain experts do not always mention bug-introducing commits or issues, which were addressed by bug-introducing commits, in the issue description or comments.

- A regular expression to extract bug-introducing commit hashes or addressed by bug-introducing commits issue IDs, cannot be universal for all projects.
- Usage of the regular expressions Issue-RE and Commit-RE may provide false positives because they are not aware of the context.

OpenSZZ version that uses issue description and comments to find bug-introducing commits on top of OpenSZZ-UIL will be called **OpenSZZ-UILDC**.

The approaches to find bug-introducing commits in OpenSZZ-UILDC are applied in the next order:

1. Using issue links.
2. Using issue description and comments.
3. Applying *git blame* to bug-fixing commits.

Each next step is executed only if the previous step did not provide results.

OpenSZZ-UIL searches for bug-introducing commits by using only approaches 1 and 3.

3.3 Improving OpenSZZ without relying on an issue tracker

This section describes how the next questions related to the "analyzing repositories without relying on an issue tracker" feature can be answered:

1. Does the feature need to be implemented?
2. How different are the results of analyzing repositories without relying on an issue tracker?

3.3.1 How often open-source repositories use Jira as an issue tracker, and it is publicly available?

An answer to this question will help to understand whether the "analyzing repositories without relying on an issue tracker" feature of OpenSZZ needs to be implemented.

Searching for projects where Jira was used as the ITS (and it is publicly available) is not trivial. A more straightforward way could be to search for needed Jira-projects (for example, where Swift is used as a programming language). Then, find repositories which use the Jira-project. The problem is that Jira is not a centralized projects storage. It is a web-accessible software that can be deployed to any server. Therefore, there is no possibility to search among all Jira projects.

The idea is to find repositories, which use Jira for issue tracking. Then, find Jira-projects corresponding to the found repositories. The way to determine if Jira is possibly used is to analyze commit messages on containing Jira-issue IDs. It is a common practice to include the Jira-issue ID in the message of the related to the issue commit.

The steps are the following:

1. Search all commits which possibly have a Jira-issue ID in their commit messages.
2. Get repositories that contain these commits.
3. Search for Jira projects based on Jira-issue ID and project name.

Search among all commits can be done using World of Code (WoC) - a prototype of an updatable and expandable infrastructure to support research and tools that rely on version control data from the entirety of open source projects [20].

3.3.2 Comparing the results of analyzing a repository with and without using Jira

The question that needs to be answered is how much different are the results of analyzing projects with the help of Jira and without it?

OpenSZZ implies using Jira. Issue opening dates are useful on the step of searching bug-introducing commits. All commits after the issue opening date can be ignored because the bug reported in the issue already was present at the moment, and the next commits could not contribute to it.

When analyzing a certain commit in a repository without Jira, we cannot know whether a bug was already present at the moment. Therefore, all commits before the bug-fixing commit will be considered as candidates to bug-introducing commits. It needs to be verified if the correct bug-introducing commits can be found without knowing when the bug was already present in the repository. The first experiment addresses this. In order to compare how different bug-introducing commits are for two different approaches in the experiment, bug-fixing commits should be the same for both of them. Therefore, the first approach involves using OpenSZZ-UILDC. The second approach involves using OpenSZZ-no-IOD.

OpenSZZ-no-IOD is a modified OpenSZZ-UILDC, where the issue opening date is not used in the bug-fixing commits search step. All commits before the bug-fixing commit can be considered as candidates to bug-introducing commits. In OpenSZZ-UILDC, all commits before the issue opening date may be considered as candidates to bug-introducing commits.

The second experiment will show how different the BFC list when BFCs are searched without using Jira.

OpenSZZ-WoJ is a modified OpenSZZ-UILDC with the difference that Jira is not used. Commits with a commit message including such keywords "fix(es), fixed, bug(s), defect(s), patch" are considered as bug-fixing commits.

Experiment 1: Search bug-introducing commits in two approaches:

- 1) OpenSZZ-UILDC.
- 2) OpenSZZ-no-IOD.

Experiment 2: Search bug-fixing commits in two approaches:

- 1) OpenSZZ-UILDC.
- 2) OpenSZZ-WoJ.

4 Results

This chapter describes the results of applying methods from section 3 and answers whether the goals G1 to G3 (defined in section 1.2) were achieved.

4.1 Evaluation of OpenSZZ correctness

This section describes such found issues with the OpenSZZ implementation that affect its results.

4.1.1 Found issues with OpenSZZ

Five issues with OpenSZZ were found. Their fixes are not improvements and do not change the intended behaviour of OpenSZZ. Without fixing them, the tool can provide incorrect and unexpected results.

Looking for bug-introducing commits:

- **Issue-1. Git blame result is not reset for different files and different commits**

  The issue is in the `getBlameAt` function of `Git` class[22]. Blame object provides a blame result for a certain file in a certain repository state (for specified commit). The file and commit are set only once and are not reset on each next `getBlameAt` call.

  The `getBlameAt` method is called for every changed line of every bug-fixing commit. Setting the blame object anew for each line can be redundant since it provides the same result for all lines of the same file on the same commit. Therefore, the blame object needs to be reset once a file or commit is changed.

- **Issue-2. Mismatched line numbers** To find commits which edited each line for the last time, OpenSZZ uses `getSourceCommit` method of the `BlameResult` class from JGit library[23]. The `getSourceCommit` method takes 0 based line number as an argument, but in OpenSZZ, a line number extracted from *git diff* result is passed as an argument. It is incorrect because in *git diff* output line numbers start with 1.

  Without a fix, for each line of a bug-fixing commit, OpenSZZ finds a commit that made the last change in the next line instead of a commit that made the last change in the current line.

[22] `https://github.com/clowee/OpenSZZ-Cloud-Native/blob/master/core/src/main/java/com/rest/szz/git/Git.java#L298`

[23] `http://archive.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/blame/BlameResult.html#getSourceCommit(int)`

- **Issue-3. Incomplete iteration of files changed by a bug-fixing commit**.

  In OpenSZZ-original, the process of iterating files of a bug-fixing commit finishes once one of the files does not have modified lines[24]. This leads to the situations when bug-introducing commits are not found because files with modified lines followed files with added new lines only[25]. Instead, file iteration should continue.

- **Issue-4. Analyzing only Java files**

  OpenSZZ-original is limited to Java projects and analyzes only `.java` files[26]. However, Java repositories contain also files of other types (for example, `.xml` and `.sh`) and bugs can be introduced with changes in such files. Instead, OpenSZZ should analyze changes in all non-binary files except of `.txt` and `.md` files (changes in text and markdown files most likely do not affect software's behaviour). This will also make it possible to use OpenSZZ with other non-Java repositories.

Looking for bug-fixing commits:

- **Issue-5. Incorrect issues retrieving**

  OpenSZZ stores issues in `.csv` files with maximum 1000 issues in each. This way, a file `<project_key>_0.csv` has issues with ID from 1 to 999, file `<project_key>_1.csv` has issues with ID from 1000 to 1999, and so on.

  To fetch certain portion of issues from Jira API, OpenSZZ uses the next parameters in JQL query:

  - `project=<project_key> ORDER BY key ASC`
  - `tempMax=1000`
  - `pager/start=<page_number*1000>`

  With `<project_key> = OOZIE` and `<page_number> = 2` the query is interpreted as follows: from all issues of OOZIE project sorted by issue key in ascending order return 1000 issues starting from 2000th result.

  On the step of linking commits to issues, OpenSZZ extracts issue IDs from commit messages. Then OpenSZZ searches the issues with IDs equal to the extracted ones not in all files with fetched issues, but only in files that are supposed to contain them. Therefore, OpenSZZ will search an issue `OOZIE-2222` in `OOZIE_2.csv`.

[24] `https://github.com/clowee/OpenSZZ-Cloud-Native/blob/master/core/src/main/java/com/rest/szz/entities/Link.java#L259`

[25] Example commit - `https://github.com/apache/oozie/commit/e0b7cde7`

[26] `https://github.com/clowee/OpenSZZ-Cloud-Native/blob/master/core/src/main/java/com/rest/szz/entities/Link.java#L254`

The process works correctly as long as issues are not deleted in the Jira project. When some issues are deleted from a Jira project, it is possible that some other issues will not be found by OpenSZZ because they are stored in another file and not in the file where they are supposed to be. For example, if any issue with the ID between 1000 and 2000 is deleted, then the query used to return 1000 results after 1000 results returns issues with IDs from 1000 to 2001. Thus, the issue with the ID 2000 will be stored in the file `<project_key>_1.csv` and will not be found in `<project_key>_2.csv`. Hence, even if a commit that references the issue with ID 2000 is a bug-fixing commit, it will not be considered bug-fixing because issues linked in the commit message will not be found.

4.1.2 Comparing results of OpenSZZ-original with OpenSZZ-corrected

Comparison of results of OpenSZZ-original and OpenSZZ-corrected on the benchmark dataset by Wen *et al.* for the Oozie project is shown in Table 9.

Columns in the table show results for different OpenSZZ versions:

1. Original - OpenSZZ-original.
2. Fix1 - OpenSZZ-original with fixed Issue-1.
3. Fix1-2 - OpenSZZ-original with fixed Issues 1-2.
4. Fix1-3 - OpenSZZ-original with fixed Issues 1-3.
5. Fix1-4 - OpenSZZ-original with fixed Issues 1-4.
6. Corrected - OpenSZZ-original with fixed Issues 1-5.

OpenSZZ-corrected finds higher number of BFCs, because without Issue-5 being fixed OpenSZZ may search issues in wrong files. Thus, higher number of commits are not linked to issues and are not considered bug-fixing.

Reasons why OpenSZZ-corrected did not find bug-introducing-commits for 2 out of 44 issues from the benchmark dataset:

- The issue was closed earlier than the bug-fixing commit was created (1 occurrence).
- No modified lines in code files (other than .md and .txt). Issues were fixed only by code additions (1 occurrence).

4.1.3 Conclusion

OpenSZZ-corrected can provide bug-introducing commits for most issues from a benchmark dataset. Cases, when bug-introducing commits are not found, are expected. It means that OpenSZZ-corrected works as intended and is ready to be improved.

Table 9. Comparison of results by OpenSZZ-original without and with applied fixes to the results from the benchmark dataset of Oozie project by Wen *et al.*

	Original	**Fix1**	**Fix1-2**	**Fix1-3**	**Fix1-4**	**Corrected**
Number of BFCs with BICs found	93/1189	675/1189	664/1189	746/1189	971/1189	993/1218
Number of BFCs from benchmark with found BICs	2/43	35/43	35/44	40/43	41/43	41/43
Sensitivity	0	0.955	0.96	0.964	0.966	0.966
Precision	0	0.833	0.835	0.833	0.839	0.839

The table can be read as follows: OpenSZZ-corrected found 1218 BFCs and found BICs for 993 out of the 1218 BFCs. The Benchmark dataset has 43 BFCs, and OpenSZZ-corrected found BICs for 41 out of the 43. The average sensitivity of the found BICs comparing them to the BICs in the benchmark dataset is 0.966, and the average precision is 0.839.

4.2 Improvement of OpenSZZ with Issue Tracker

This section presents the results of improving OpenSZZ by addressing the limitation of B-SZZ - "Bug-fixing commit may contain changes not related to fixing a bug". The result of implementing each of the following steps are presented:

1. Ignoring whitespace changes.
2. Ignoring changes in comment lines.
3. Ignoring formatting and comment changes in Java files.
4. Ignoring refactoring changes.
5. Searching for bug-introducing commits based on information provided by domain experts in Jira issues.

4.2.1 Ignoring whitespace changes

OpenSZZ-IWS (an OpenSZZ version that ignores trailing whitespace changes and blank line changes) was applied to the Oozie project. The comparison of results by OpenSZZ-IWS to the results from the benchmark dataset of the Oozie project by Wen *et al.* is shown in Table 10.

Comparing results to the benchmark dataset does not show a difference between OpenSZZ-corrected and OpenSZZ-IWS. However, OpenSZZ-IWS found BICs for 15 BFCs less than OpenSZZ-corrected - 978 instead of 993. In all the 15 BFCs, a bug is

fixed with only line additions and the whitespace changes in existing lines are unrelated to the bugs.

OpenSZZ-IWS is expected to provide better results than OpenSZZ-corrected because it ignores whitespace changes that could not influence software's behaviour.

4.2.2 Ignoring changes in comment lines

OpenSZZ-IC (Ignore comments) is OpenSZZ-IWS that ignores changes in comment lines.

The comparison of the results by OpenSZZ-IC to the results from the benchmark dataset of the Oozie project by Wen *et al.* is shown in Table 10.

Comparing results to the benchmark dataset does not show a difference between OpenSZZ-IWS and OpenSZZ-IC. However, OpenSZZ-IWS found BICs for 13 BFCs less than OpenSZZ-IWS - 965 instead of 978.

The 13 BFCs were reviewed:

- 4/13 BFCs indeed addressed bugs related to comments (2 of them are about code style in Javadoc comments[27]).
- 1/13 BFCs had a change in Javadoc comments which does not fix a bug by itself but is related to the new code lines added that fix the bug.
- In 2/13 BFCs, bugs were fixed by uncommenting commented out code lines.

Williams and Spacco proposed to ignore changes in comments [10]. Neto, Costa and Kulesza ignored lines that have Java-style comments in their SZZ implementations (AG-SZZ, MA-SZZ, R-SZZ, L-SZZ) [3].

However, it was found that ignoring changes in comments is not always a good idea. Therefore, OpenSZZ will have the feature of ignoring changes in comment lines optional.

4.2.3 Ignoring formatting and comment changes in Java files

OpenSZZ-DJ is an OpenSZZ-corrected that uses the DiffJ tool to mine code differences made by bug-fixing commits. DiffJ is supposed to ignore formatting changes and changes in comments.

The OpenSZZ-DJ result is compared to the results of OpenSZZ-IC and OpenSZZ-Corrected in Table 10.

[27] `https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html`

Table 10. Comparison of results by OpenSZZ-corrected, OpenSZZ-IWS, OpenSZZ-IC and OpenSZZ-DJ to the results from the benchmark dataset of Oozie project by Wen *et al.*

	Corrected	**IWS**	**IC**	**DJ**
Number of BFCs with BICs found	993/1218	978/1218	965/1218	926/1218
Number of BFCs from benchmark with found BICs	41/43	41/43	41/43	39/43
Sensitivity	0.966	0.966	0.966	0.964
Precision	0.839	0.839	0.839	0.79

The table can be read as follows: OpenSZZ-IC found 1218 BFCs and found BICs for 965 out of the 1218 BFCs. The Benchmark dataset has 43 BFCs, and OpenSZZ-corrected found BICs for 41 out of the 43. The average sensitivity of the found BICs comparing them to the BICs in the benchmark dataset is 0.966, and the average precision is 0.839.

Issues with using DiffJ for mining code changes Results of analyzing Oozie repository with OpenSZZ-DJ were reviewed, and the next issues with DiffJ were found:

- Since DiffJ operates with code and not with lines, it considers a code inserted into existing code lines as added code and not as changed. Example of a code difference interpreted as code addition by DiffJ is shown in Figure 10. DiffJ does not distinguish between additions within existing code lines and additions of code as new lines. Therefore, if OpenSZZ traces back only lines with changed or deleted code, such additions within existing code lines as shown in Figure 10 are missed.
- DiffJ does not support Java 7 and greater syntax[28]. The tool fails to extract changes when unsupported syntax is found.
- Some changes cannot be detected, for example, changes in explicit constructor invocations[29]
- DiffJ compares code without taking code location into consideration, what causes differences to be found where there are no differences[30].

DiffJ may have other not reported issues. They are unlikely to be fixed in the nearest future because the repository does not have changes after the last release in 2018 and seems to be not maintained.

Due to the issues found, using OpenSZZ-DJ to mine code differences is discouraged. The default approach to mine code changes with *git diff* will be used for Java files as well as for other files.

[28] `https://github.com/jpace/diffj/issues/3`
[29] `https://github.com/jpace/diffj/issues/7`
[30] `https://github.com/jpace/diffj/issues/8`

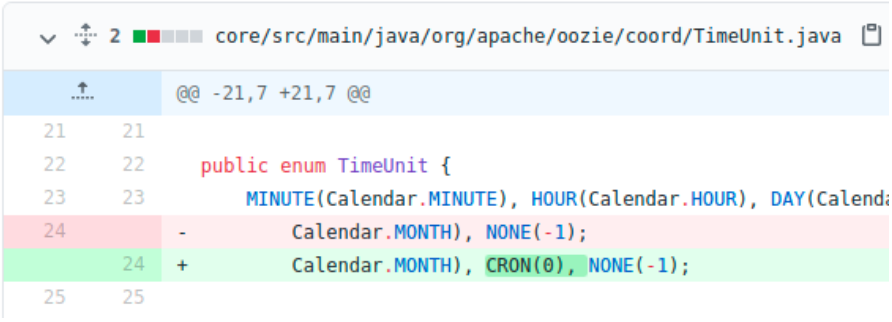


Figure 10. A code difference that is interpreted as addition by DiffJ and not as a change.

4.2.4 Ignoring refactoring changes in Java files

OpenSZZ-RA (Refactoring aware) is OpenSZZ-IWS that ignores refactoring changes in bug-fixing commits.

The comparison of results by OpenSZZ-IC and OpenSZZ-RA to the results from the benchmark dataset of the Oozie project by Wen *et al.* is shown in Table 11.

Table 11. Comparison of results by OpenSZZ-IC and OpenSZZ-RA to the results from benchmark dataset of Oozie project by Wen *et al.*

	IC	**RA**
Number of BFCs with BICs found	965/1218	953/1218
Number of BFCs from benchmark with found BICs	41/43	41/43
Sensitivity	0.966	0.966
Precision	0.839	0.839

The table can be read as follows: OpenSZZ-RA found 1218 BFCs and found BICs for 953 out of the 1218 BFCs. The Benchmark dataset has 43 BFCs, and OpenSZZ-corrected found BICs for 41 out of the 43. The average sensitivity of the found BICs comparing them to the BICs in the benchmark dataset is 0.966, and the average precision is 0.839.

The difference between the results of analyzing the Oozie project by OpenSZZ-IC and OpenSZZ-RA is not visible by comparison the results to the benchmark dataset of the Oozie project by Wen *et al.*

OpenSZZ-RA found BICs for 12 BFCs less than OpenSZZ-IC. These BFCs were reviewed. OpenSZZ-RA could not find BICs, because in all the BFCs bugs were fixed with lines addition and refactoring changes are the only changes in existing code lines.

4.2.5 Searching for bug-introducing commits based on information provided by domain experts in Jira issues

OpenSZZ-UILDC searches for bug-introducing commits with the next steps:

1. Search BICs using issue links.
2. If BICs are not found with step 1, then search BICs using issue description and comments.
3. If BICs are not found with step 2, then search BICs with default approach - using *git blame* feature.

OpenSZZ-UILDC found 1218 bug-fixing commits in the Oozie repository. Figure 11 shows the distribution of sources of how bug-introducing commits for the bug-fixing commits of the Oozie repository were found with OpenSZZ-UILDC.

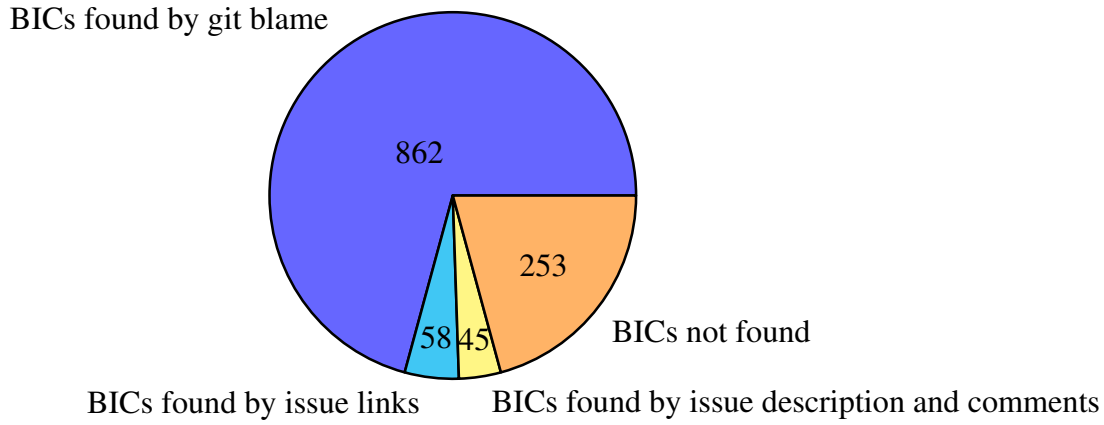


Figure 11. Distribution of sources how BICs of Oozie repository were found with OpenSZZ-UILDC.

The results for OpenSZZ-UIL and OpenSZZ-UILDC are shown in Table 12. They are expected because the OpenSZZ versions used the same approach to find bug-introducing commits, which Wen *et al.* used to create their benchmark dataset. OpenSZZ-UILDC found BICs for 12 BFCs more than OpenSZZ-RA. OpenSZZ-RA did not find them, because bugs were fixed with only line additions in all the BFCs.

Using issue links, description, and comments for searching for bug-introducing commits will be optional.

Table 12. Comparison of results by OpenSZZ-RA, OpenSZZ-UIL, and OpenSZZ-UILDC to the results from the benchmark dataset of Oozie project by Wen *et al.*

	RA	**UIL**	**UILDC**
Number of BFCs with BICs found	953/1218	957/1218	965/1218
Number of BFCs from benchmark with found BICs	41/43	42/43	42/43
Sensitivity	0.966	0.988	0.988
Precision	0.839	0.917	0.901

The table can be read as follows: OpenSZZ-UILDC found 1218 BFCs and found BICs for 965 out of the 1218 BFCs. The Benchmark dataset has 43 BFCs, and OpenSZZ-UILDC found BICs for 42 out of the 43. The average sensitivity of the found BICs comparing them to the BICs in the benchmark dataset is 0.988, and the average precision is 0.901.

4.2.6 Conclusion

OpenSZZ was improved on the step of searching bug-introducing commits.

Bug-introducing commits are searched based on changes in bug-fixing commits. Bug-fixing commits can contain changes that do not influence software's behaviour and are not related to the bug fixing.

OpenSZZ ignores the next types of changes in the code of a bug-fixing commit:

1. Whitespace changes.
2. Changes in comment lines (optional).
3. Refactoring changes in Java code.

Also, OpenSZZ allows searching for bug-introducing commits based on information provided by domain experts in Jira issues.

4.3 Improving OpenSZZ without relying on an Issue Tracker

This section answers whether the "analyzing repositories without relying on an issue tracker" feature needs to be implemented in OpenSZZ and how different are the results of analyzing repositories using the feature?

4.3.1 How often open-source repositories use Jira as an issue tracker, and it is publicly available?

The task was to find out how often Jira is used for Swift projects. A script for World of Code to find projects which possibly use Jira was written[31]. It provides a list of tuples: repository name (with author), issue ID used in at least one of its commits, the link to a GitHub repository with this commit (if the repository was found on GitHub).

As a result, 256 such projects, where Jira is possibly used, were found with WoC among 66316 repositories with Swift code stored on the WoC server. Attempts to search for Jira-projects (it was performed manually) corresponding to the repositories were unsuccessful. Only three Jira projects corresponding to GitHub repositories were found from the first 70 projects with search queries "<Jira_key> + Jira" and "<project_name> + <Jira_key> + Jira".

Most of the SZZ-related studies analyze Java projects because the Jira issue tracker is more likely to be used with them. GitHub Issues is common as an issue tracker among modern open-source projects, which are hosted on GitHub. But there are reasons why SZZ is usually applied to the projects, where Jira is used as an issue tracker:

- Jira is used usually in the same way across different projects. It proposes lists of default issue types, statuses, and resolutions. There is a chance that developers of different projects will assign the *Bug* type to different kinds of issues. Or that they will use custom resolution values instead of *Fixed*. But the way how issues are handled on GitHub Issues across different repositories may differ much more (status can be only *open* or *closed*, all labels are custom).
- It is not common to include the ID of the addressed GitHub issue in the commit message. Commit message usually has the pull request ID. The addressed issue's ID is often mentioned in the pull request's description. Thus, linking bug-fixing commits to the bug-reporting issues may be more complicated than with Jira.

The number of repositories that use publicly available Jira as an issue tracker is limited. Therefore, a way to analyze repositories without relying on Jira issue tracker needs to be implemented.

4.3.2 Comparing results of analyzing a repository with and without using Jira

A repository to be analyzed is Oozie repository by The Apache Software Foundation[32]. The repository was used for evaluation of different OpenSZZ versions in sections 4.1 and 4.2.

[31] `https://github.com/VladyslavBondarenko/OpenSZZ-replication/tree/master/scripts/WoC`

[32] OOZIE repository - `https://github.com/apache/oozie`, Jira - `https://issues.apache.org/jira/projects/OOZIE`

Experiment 1

- 1) Using the issue opening date: all commits before the issue opening date may be considered candidates to bug-introducing commits (OpenSZZ-UILDC).
- 2)Without using the issue opening date: all commits before the bug-fixing commit may be considered candidates to bug-introducing commits (OpenSZZ-no-IOD).

OpenSZZ-UILDC found BICs for 965/1218 BFCs. OpenSZZ-no-IOD found BICs for 1004/1218 BFCs, which is 39 BICs more.

OpenSZZ-no-IOD found more BICs, because commits that edited changed by a bug-fixing commit lines after the issue opening are considered as candidates to bug-introducing commits.

There are the next reasons why bug-fixing commits change lines edited after issue opening:

- A bug was reported for being present in a certain file, and identical bugs were introduced in other files after opening the issue.
- The changes are formatting changes.
- The changes are refactoring changes.
- The changes are TODO-statements added after issue opening with comments to the lines with bugs).

Experiment 2

- 1) Using Jira to identify bug-fixing and bug-introducing commits (OpenSZZ-UILDC).
- 2) Without using Jira (OpenSZZ-WoJ).

OpenSZZ-UILDC found bug-introducing commits for 965 bug-fixing commits, OpenSZZ-WoJ - for 166.

129/166 bug-fixing commits from OpenSZZ-WoJ results have an issue ID in the commit message. But only 71 of them reference issues with the *Bug* type.

The comparison of results by OpenSZZ-WoJ and OpenSZZ-UILDC to the results from the benchmark dataset of the Oozie project by Wen *et al.* is shown in Table 13.

Noticed that different commits may have the same commit message, for example ("fix docs", "fix typo", "fix eclipse warnings", "javadoc fixes").

Table 13. Comparison of results by OpenSZZ-WoJ and OpenSZZ-UILDC to the results from benchmark dataset of Oozie project by Wen *et al.*

	UILDC	**WoJ**
Number of BFCs with BICs found	965/1218	166/234
Number of BFCs from benchmark with found BICs	42/43	3/43
Sensitivity	0.988	1
Precision	0.901	0.722

The table can be read as follows: OpenSZZ-WoJ found 234 BFCs and found BICs for 166 out of the 234 BFCs. The Benchmark dataset has 43 BFCs, and OpenSZZ-WoJ found BICs for 3 out of the 43. The average sensitivity of the found BICs comparing them to the BICs in the benchmark dataset is 1, and the average precision is 0.722.

4.3.3 Conclusion

It is better to use Jira when possible. Although the approach without Jira provides more results - many of them can be false positives.

It is better to use the date of issue opening while searching for bug-introducing commits. Because usually, changes in the lines made after issue opening are not related to the bug fixing.

4.4 OpenSZZ improvements not related to the SZZ algorithm itself

OpenSZZ is a Java application that can be built into Jar file and used via a command-line interface. OpenSZZ-Cloud-Native is a wrapper around OpenSZZ containing graphic interface and scheduler services in addition to the OpenSZZ itself. Each service is packaged into a docker-container. This way, OpenSZZ-Cloud-Native can run in a virtualized environment independently from the local environment. OpenSZZ-Cloud-Native has a docker-compose file that contains configurations for all its services. This allows to create and run them with a single command.

In this section, OpenSZZ-original will imply OpenSZZ-Cloud-Native original, OpenSZZ-improved will imply OpenSZZ-Cloud-Native with all improvements done in the thesis.

4.4.1 Fetching Jira issues and linking them to commits

OpenSZZ stores fetched Jira issues in text files instead of a database. OpenSZZ saves fetched issues in multiple files instead of a single file because some projects may have a lot of issues, and the file could be too big for operating with it.

Transforming Jira URL into Jira API URL to fetch issues.

OpenSZZ-original was used only for analysing projects of Apache Software Foundation. All Jira projects of Apache Software Foundation have the next URL - `https://issues.apache.org/jira/projects/<project_key>` (Jira URL). An API URL to fetch issues looks this way for the projects - `https://issues.apache.org/jira/sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml` (Jira API URL). OpenSZZ-original transforms Jira URL into Jira API URL by appending `/jira/sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml` to the part of Jira URL before `/jira/projects/`. It works correctly for projects from Apache Software Foundation but not for other Jira projects.

For example, when Jira URL is `https://jira.catrob.at/projects/CATTY`, OpenSZZ-original will not be able to combine Jira API URL because Jira URL does not include `/jira/projects/`.

Solution: transform Jira URL into Jira API URL by appending `/sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml` to the part of Jira URL before `/projects/`.

Fetching Jira issues.

OpenSZZ-original fetches issues of any type and links commits to them. OpenSZZ-improved uses only issues with the *Bug* type to identify bug-fixing commits. Since issues of other types are unused, only issues with the *Bug* type are fetched and stored by OpenSZZ-improved.

Storing issue comments.

Although issue comments are fetched and saved for each issue by OpenSZZ-original, they are unused. All comments are fetched together as a single string but written into a file that many times as there are comments. Authors assumed that comments are fetched one by one. The issue does not influence the correctness of OpenSZZ but impacts its performance. The process of retrieving Jira issues is faster when the issue is solved.

4.4.2 Reusing working files from the previous analysis

Before executing the SZZ algorithm, OpenSZZ fetches Jira issues and saves them into text files, clones git-repository and saves commits from the repository into a text file.

OpenSZZ-original deletes all working files right after the end of project analysis. Thus, every time Jira issues are fetched, git-repository is cloned and commits from the repository are retrieved. It takes time.

OpenSZZ-improved has an option to reuse the working files from the previous analysis of the project. Having this option enabled not only facilitates the analysis process but also allows to test different OpenSZZ configurations with a guarantee that commits and Jira issues remain the same.

OpenSZZ-improved uses existing working files and does not delete them once the analysis is completed. Working files are saved in separate directories for each project. If the *Use working files from previous analysis* checkbox is unchecked (default state), the project working directory is cleaned before starting analysis and Jira issues and repository commits are fetched anew.

4.4.3 Identifying bug-fixing and bug-introducing commits

A memory leak was found and fixed.

OpenSZZ reads git-repository data on the preparation step saving all repository commits into a text file and on the step of searching for bug-introducing commits with the help of JGit library[33]. `Git.open` method of the `Git` class takes a directory path as input and returns a `Git` object [34] representing a git-repository in the directory.

The problem is that the `Git` objects were not cleaned after usage and remained in the Java heap memory.

4.4.4 OpenSZZ output

OpenSZZ-original has the next columns in the result output:

- bugFixingId
- bugFixingTs
- bugFixingfileChanged
- bugInducingId
- bugInducingTs
- issueType

Column *bugFixingTs* has date values in the next format: `yyyy-MM-dd'T'HH:mm:ss.SSSZ`. The format was changed to `yyyy-MM-dd'T'HH:mm:ssZ` to remove milliseconds from values, because git commits have date of creation stored in seconds. Thus, milliseconds in the values are always 0.

Column *issueType* is removed because bug-fixing commits are linked only to issues with the *Bug* type.

Column *issueId* was inserted to show to which issue is bug-fixing commit linked.

Column *note* was inserted to show notes about bug-introducing commits. By default, the value is empty, but it can contain the next values:

[33] `https://www.eclipse.org/jgit/`

[34] `http://archive.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/api/Git.html`

- "Ignored file type" or "No changed lines, only additions" - reasons why a bug-introducing commit was not found for the file.
- "brokenBy" - bug-introducing commit was found by linking the commit to an issue referenced by the "brokenBy" type of link in the bug-reporting issue.
- "description/comments" - bug-introducing commit was found by linking the commit to an issue referenced in the bug-reporting issue description or comments.

OpenSZZ-original inserts a blank line after each line in the output result. It was changed to avoid blank lines in the output result.

4.4.5 OpenSZZ-Cloud-Native interface changes

OpenSZZ-Cloud-Native is a wrapper for the OpenSZZ application that allows to deploy it as a web application. OpenSZZ-Cloud-Native provides a graphical interface for OpenSZZ. The section describes changes made in the interface corresponding to the improvements made in OpenSZZ.

To use OpenSZZ for analysing repositories both with using issue tracker and without relying on the issue tracker, the next changes were made in the initial user interface:

- *Use Jira* checkbox was added. Value of the checkbox defines which field is shown and is required - *Jira URL* or *Search query*. A repository is analysed according to the chosen option.
- A *Search query* field was added. The field is hidden when the *Use Jira* checkbox is checked and is shown and is required when the checkbox is unchecked. The field takes a regular expression as a value. The regular expression is applied to commit messages to filter specific commits.
- *Jira URL* field is shown and is required when the *Use Jira* checkbox is checked and is hidden when the checkbox is unchecked.
- The third line of Example values shows either example value for the *Jira URL* field or example value for *Search query* field (depending on the state of *Use Jira* checkbox.

Other changes that were done in the OpenSZZ user interface:

- *Project Name* field is not required anymore. Value of the field is used on the *ANALYZED PROJECTS* page. If not, the set repository name will be used by default.

- *Email* field is not required anymore. Value of the field defines email where the result of the analysis will be sent.
- *Ignore changes in comment lines* checkbox was added (unchecked by default).
- *Add all BFC to result* checkbox was added (unchecked by default). With unchecked state, only bug-fixing commits and corresponding bug-introducing commits are added to the result file. When the checkbox is checked, bug-fixing commits without bug-introducing commits found are added to the result file. This option allows knowing bug-fixing commits for which bug-introducing commits were not found.
- *Use working files from previous analysis* checkbox was added (unchecked by default). Working files are cloned git-repository, files with fetched Jira issues, a file with git-repository commits. When the option is enabled, OpenSZZ will reuse the files. It will speed up the analysis (by skipping the preparations). The option also is helpful for testing different OpenSZZ options with a guarantee that commits and Jira issues remain the same. *Project Name* field can be filled in with a unique value to ensure that working files will not be deleted by running another analysis for the same repository with disabled *Use working files from previous analysis* option.

The original interface of OpenSZZ-Cloud-Native is shown in Figure 12. The improved version of the interface is shown in Figure 13 and Figure 14.

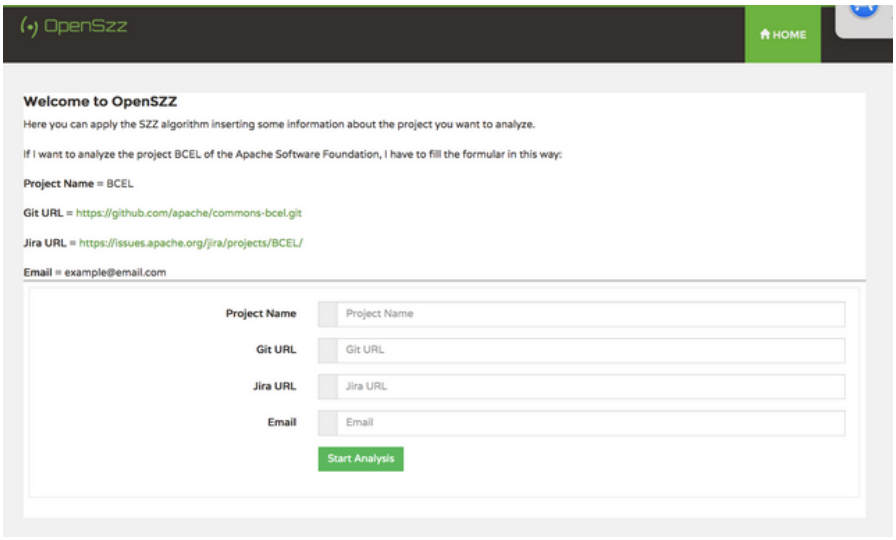


Figure 12. OpenSZZ - Cloud Native. Original interface.

All form fields are described to familiarize a user with all OpenSZZ options. Figure 15 show the part of interface with the form fields description.

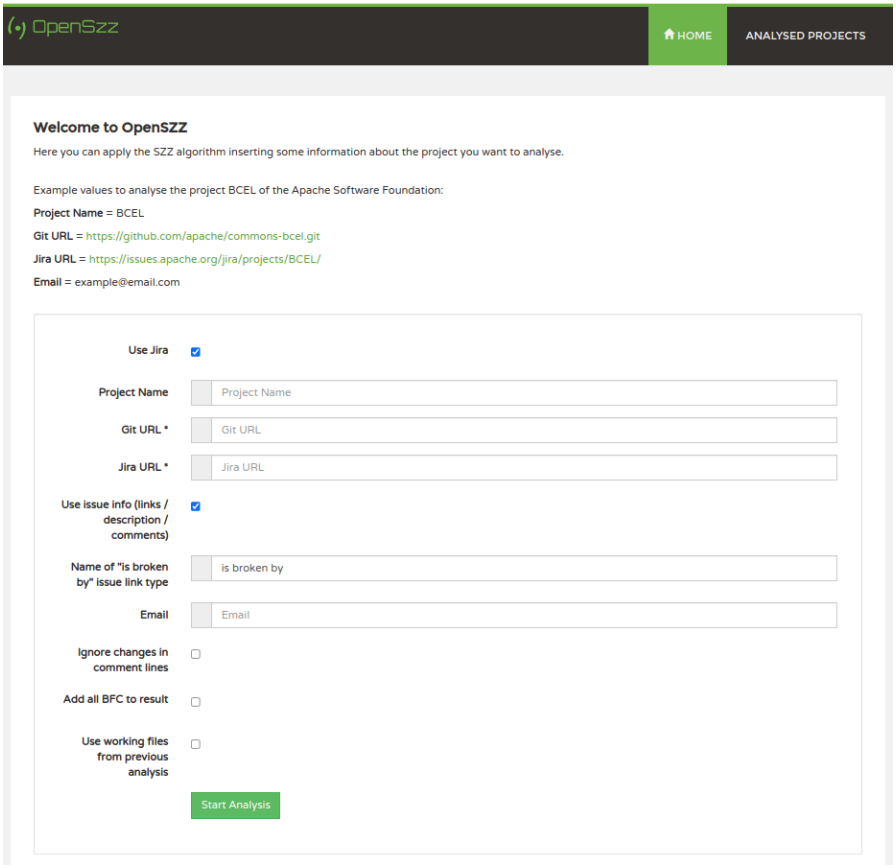


Figure 13. OpenSZZ - Cloud Native. Improved interface (with *Use Jira* checked).

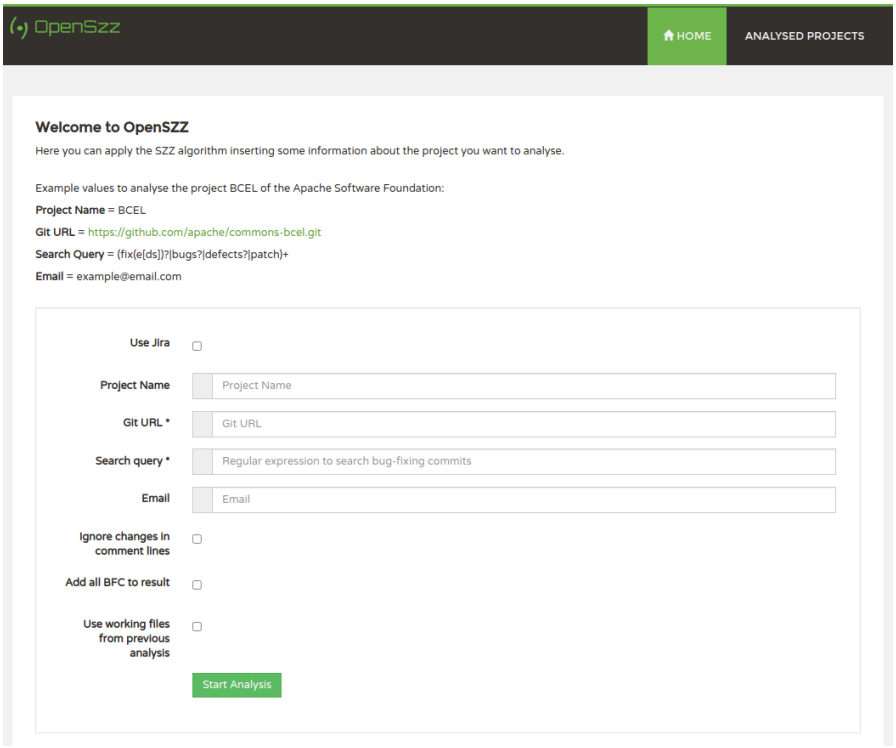


Figure 14. OpenSZZ - Cloud Native. Improved interface (with *Use Jira* unchecked).

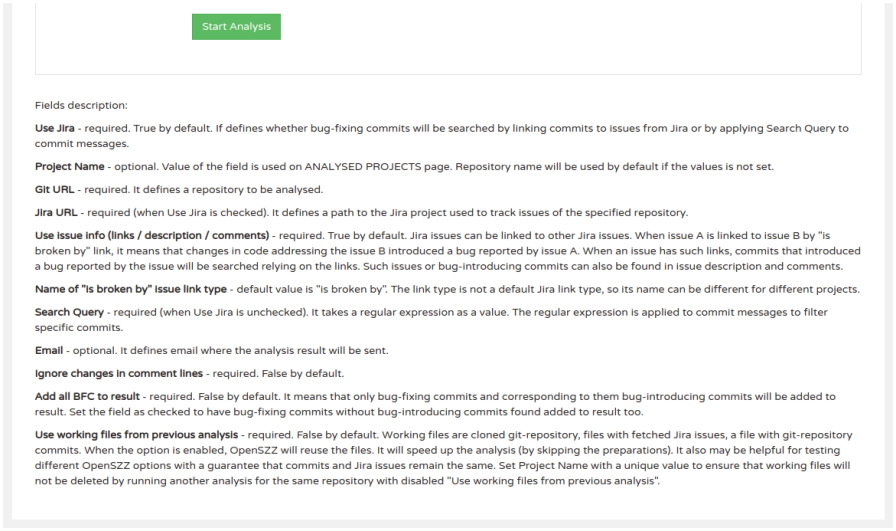


Figure 15. OpenSZZ - Cloud Native. Improved interface - fields description.

5 Conclusion

This chapter describes achieved goals and compares OpenSZZ improved in this thesis to other SZZ implementations.

5.1 Achieved goals

This section briefly describes results related to the goals G1 to G3 defined in section 1.2.

G1. Evaluate the correctness of OpenSZZ

Issues with the OpenSZZ implementation were found. Five of them cause results to be incorrect. The issues were resolved.

G2. Improve OpenSZZ - apply improvements proposed by other studies and overcome limitations.

OpenSZZ considers such commits bug-fixing that have referenced a bug-reporting issue in their commit message. Then OpenSZZ searches bug-introducing commits for each file modified in each bug-fixing commit. The most recent commit that edited any of the lines edited later by the bug-fixing commit is considered a bug-introducing commit (with the condition that the commit was created before the issue was opened).

To identify bug-introducing commits more accurately, OpenSZZ ignores:

- Whitespace changes.
- Changes in comment lines (optional).
- Refactoring changes in Java files.

OpenSZZ does not trace back the lines to find when they were edited for the last time before a bug-fixing commit. Consequently, a part of deliberately false candidates for bug-introducing commits is excluded, and chances to identify a bug-introducing commit correctly increase.

OpenSZZ provides an option to use information in issues (issue links, description and comments) to search bug-introducing commits.

G3. Improve OpenSZZ - make it possible to use it for repositories without ITS used

OpenSZZ-Cloud-Native has an option to analyse repositories without an issue tracker. In this case, commits are considered bug-fixing based only on their commit message.

5.2 Comparison of OpenSZZ-improved to other SZZ implementations

This section provides a brief comparison of OpenSZZ-improved (an OpenSZZ version improved in this thesis) to other SZZ implementations.

SZZ-implementations used for comparison:

- R-SZZ - an SZZ version proposed by Neto, Costa and Kulesza [3] and reimplemented by Rosa *et al.* [21].
- RA-SZZ* - improved RA-SZZ proposed by Neto, Costa and Kulesza [17].
- SZZ-Unleashed - an SZZ implementation proposed by Borg *et al.* [4].
- OpenSZZ - OpenSZZ-Cloud-Native presented by Lenarduzzi *et al.* [2].
- OpenSZZ-improved - OpenSZZ-Cloud-Native improved in this thesis.

SZZ-implementations not used for comparison:

- B-SZZ - the basic version of OpenSZZ.
- AG-SZZ, MA-SZZ, L-SZZ - the SZZ versions were implemented by Neto, Costa and Kulesza [3] and reimplemented by Rosa *et al.* [21]. R-SZZ is built on top of AG-SZZ and MA-SZZ. L-SZZ is built on top of AG-SZZ and MA-SZZ as well, but R-SZZ is claimed to provide better results [3].
- RA-SZZ - an implementation of a refactoring-aware version of SZZ proposed by Neto, Costa and Kulesza [3]. RA-SZZ* will be used instead as an improved version of RA-SZZ.
- PyDriller - a general-purpose tool for analyzing git repositories. PyDriller has a method that takes a commit as input and maps files changed in this commit to commits that last modified the files before. The tool is not used for comparison because it implements only the second part of the SZZ algorithm and is not positioned as a complete SZZ implementation.

A comparison of different SZZ implementations is presented in Table 14.

OpenSZZ-improved adopts all the improvements present in other implementations of the SZZ algorithm. In addition, it provides an option to search bug-introducing commits based on information present in an issue tracker and provides an option to find bug-fixing commits without an issue tracker. OpenSZZ can be deployed as a web application with a graphical user interface and can be scaled to perform multiple repository analyses in parallel.

Table 14. Comparison of different SZZ implementations.

✓- the option is implemented. • - the option is implemented, but disabled.

	R-SZZ	**RA-SZZ***	**SZZ-Unleashed**	**OpenSZZ**	**OpenSZZ-improved**
Identify BFCs			✓	✓	✓
Identify BICs in Java repositories	✓	✓	✓	✓	✓
Identify BICs in repositories other than Java	✓		✓	•	✓
Ignore all non-executable changes in Java files		✓			•
Ignore refactoring changes in Java files		✓			✓
Ignore whitespace changes not only in Java files	✓				✓
Ignore comment changes not only in Java files	✓				✓
Analyze multiple projects in parallel				✓	✓
Deploy as a web-application				✓	✓
Identify BICs using information from an issue tracker					✓
Identify BFC and BIC without an issue tracker					✓

6 Future work

The next tasks can be done to improve OpenSZZ further:

- **Create complete benchmark datasets and verify OpenSZZ results**

  Complete benchmark datasets for evaluation SZZ implementations need to be created. Complete dataset should contain all bug-fixing commits with corresponding bug-introducing commits for a certain repository, not only few of them as it is in benchmark datasets used in other studies.

- **Analyze bugs fixed by inserting new code lines without changes in existing code lines.**

  SZZ algorithm analyzes changed code lines to find bug-introducing commits. However, bugs can be fixed by inserting new code lines without changes in existing code lines.

  For example, OpenSZZ found 1218 BFC for the Oozie repository. BICs were not found for 253/1218 BFC. For 179/253 BFCs, BICs were not found because files changed in BFC did not have lines changed, only new lines added. It is 7% of all BFCs in Oozie repository.

  The SZZ algorithm is not intended to handle such cases. Thus, a way to handle them needs to be found.

- **Support projects with closed source and issue tracking systems other than Jira.** Currently OpenSZZ supports only Jira, which is not the only option used among developers. Also, support for projects with closed source or/and non-publicly available issue tracker used may be needed in the future.

- **Revisit the way of ignoring non-executable changes.**
  Tools for mining code changes ignoring formatting changes and tools for mining refactoring changes for different programming languages need to be found and integrated into OpenSZZ.

- **Revisit a way to choose correct bug-introducing commits among multiple suspects.**
  In the proposed implementation, OpenSZZ provides a single bug-introducing commit for each file changed in the bug-fixing commit. But some changed files can be not related to the bug. Or vice-versa, a bug can be introduced with a combination of multiple commits in single file.

References

[1] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, May 2005.

[2] Valentina Lenarduzzi, Fabio Palomba, Davide Taibi, and Damian Andrew Tamburri. *OpenSZZ: A Free, Open-Source, Web-Accessible Implementation of the SZZ Algorithm*, page 446–450. Association for Computing Machinery, New York, NY, USA, 2020.

[3] E. C. Neto, D. A. da Costa, and U. Kulesza. The impact of refactoring changes on the szz algorithm: An empirical study. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 380–390, 2018.

[4] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. Szz unleashed: An open implementation of the szz algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, MaLTeSQuE 2019, page 7–12, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Valentina Lenarduzzi, Nyyti Saarimäki, and Davide Taibi. The technical debt dataset. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE'19, page 2–11, New York, NY, USA, 2019. Association for Computing Machinery.

[6] S. Kim, T. Zimmermann, K. Pan, and E. J. Jr. Whitehead. Automatic identification of bug-introducing changes. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 81–90, 2006.

[7] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017.

[8] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced? bias in bug-fix datasets. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, page 121–130, New York, NY, USA, 2009. Association for Computing Machinery.

[9] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère. Empirical evaluation of bug linking. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 89–98, 2013.

[10] Chadd Williams and Jaime Spacco. Szz revisited: Verifying when changes induce fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, DEFECTS '08, page 32–36, New York, NY, USA, 2008. Association for Computing Machinery.

[11] Gema Rodríguez-Pérez, Gregorio Robles, Alexandr Serebrenik, Andy Zaidman, Daniel Germán, and Jesus Gonzalez-Barahona. How bugs are born: a model to identify how bugs are introduced in software components. *Empir Software Eng 25*, 2020.

[12] R. Souza, C. Chavez, and R. A. Bittencourt. Rapid releases and patch backouts: A software analytics approach. *IEEE Software*, 32(2):89–96, 2015.

[13] Ming Wen, Rongxin Wu, Yepang Liu, Yongqiang Tian, Xuan Xie, Shing-Chi Cheung, and Zhendong Su. Exploring and exploiting the correlations between bug-inducing and bug-fixing commits. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 326–337, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Thomas Zimmermann, Sunghun Kim, Andreas Zeller, and E. James Whitehead. Mining version archives for co-changed lines. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, page 72–75, New York, NY, USA, 2006. Association for Computing Machinery.

[15] D. Silva and M. T. Valente. Refdiff: Detecting refactorings in version histories. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 269–279, 2017.

[16] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, and Danny Dig. Accurate and efficient refactoring detection in commit history. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 483–494, New York, NY, USA, 2018. ACM.

[17] E. C. Neto, D. A. d. Costa, and U. Kulesza. Revisiting and improving szz implementations. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2019.

[18] Jeff Pace. Diffj - a tool that compares java files based on their code, without regard to formatting, organization, comments, or whitespace changes. `http://www.incava.org/projects/java/diffj`,. 2007.

[19] Steven Davies, Marc Roper, and Murray Wood. Comparing text-based and dependence-based approaches for determining the origins of bugs. *Journal of Software: Evolution and Process*, 26, 01 2014.

[20] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus. World of code: An infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154, 2019.

[21] Giovanni Rosa, Luca Pascarella, Simone Scalabrino, Rosalia Tufano, Gabriele Bavota, Michele Lanza, and Rocco Oliveto. Evaluating szz implementations through a developer-informed oracle. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, page To Appear, 2021.

Appendix

I. Glossary

Software bug - an error, flaw, or fault in a computer program or system that causes it to produce an incorrect or unexpected result or to behave in unintended ways.

Cloud-native application - an application built as a set of microservices that run in Docker containers. The containers package all software needed to execute the application into one executable package and run in a virtualized environment. The application can be deployed via declarative code.

Version control system (VCS) - a software tool that helps in recording changes made to files by keeping a track of modifications done to the code.

Git - a free and open source distributed version control system.

Git commit - an operation which sends the latest changes to the source code to the repository. A commit contains a snapshot of the project, hashes of parent commits if any, the author/committer information (name, email and timestamp) and a commit message.

Annotation/blame feature of a VCS - a feature that shows what revision and author last modified each line of a given file. The command was renamed from "annotate" to "blame" in Git VCS.

Issue tracking system (ITS) - a computer software package that manages and maintains lists of issues.

Jira - an issue and project tracking software by Atlassian.

Just-In-Time (JIT) defect prediction - a classification model that is trained using historical data to predict bug-introducing changes.

Regular expression - a sequence of characters that specifies a search pattern. Each character in a regular expression (that is, each character in the string describing its pattern) is either a metacharacter, having a special meaning, or a regular character that has a literal meaning.

Code refactoring - a process of restructuring existing code without changing its external behavior. Refactoring is intended to improve the design, structure, and/or implementation of the software (its non-functional attributes), while preserving its functionality.

II. Replication package

A fork of the OpenSZZ repository - https://github.com/VladyslavBondarenko/OpenSZZ.

A fork of the OpenSZZ-Cloud-Native repository - https://github.com/VladyslavBondarenko/OpenSZZ-Cloud-Native.

A tool used for comparison different OpenSZZ versions to benchmark datasets - https://github.com/VladyslavBondarenko/OpenSZZ-evaluation.

Artifacts of analyzing Oozie project with different OpenSZZ versions used in sections 4.1 and 4.2 - https://github.com/VladyslavBondarenko/OpenSZZ-replication.

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Vladyslav Bondarenko**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   OpenSZZ - evaluation and improvement,

   supervised by Dietmar Pfahl.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Vladyslav Bondarenko
13/05/2021