BACHELOR THESIS

Exploring Architectural Relationships Between Issues and Email Threads in Open-Source Systems using Cosine Similarity

Martijn Kruijer

Supervisors: dr. M. Soliman and prof. dr. ir. P. Avgeriou

August 25, 2023

Abstract

To prevent future problems when developing or maintaining software, it is crucial to have architectural knowledge (AK) documented. A problem that can occur is that the documentation is scattered between multiple places. Two notable places where AK is currently stored are mailing lists and issue-tracking systems. There currently is no defined relationship between emails and issues, making it hard to find related issue-mail pairs where AK is stored.

In this study we attempt to find and define these relationships, as well as provide a way to find them using cosine similarity. For this we used 43815 emails and 70759 issues from Apache projects. From this we identified six occurring patterns and their characteristics. We also used different types of cosine similarity and determined their precision for finding emails and issues that talk about the same architectural design decision.

Keywords: Architectural design decisions, Architectural knowledge, Cosine similarity, Issue tracking systems, Mailing lists

1 INTRODUCTION

Managing architectural knowledge has been an important topic in software development [2, 17] and the need to explicitly document architectural knowledge has been emphasized both in research and in industry [1, 3]. Architectural knowledge, as defined by Kruchten et al[9], states that architectural knowledge consists of (architectural) design decisions and design. Losing these architectural design decisions (ADDs) can lead to several problems such as violating rules and constrains, or obsolete decisions not being removed[8]. This reaffirms the need for proper documentation of ADDs.

Part of good documentation is that you can find it later with relative ease. If important information is scattered, it can be difficult to locate, and there is a chance that some crucial information may go unnoticed. Two notable ways to document and share these design decisions are mailing lists[2, 12] and issue trackers[1, 16], with Mannan et al showing that 89.51% of all design discussions occur in project mailing list[11]. There exists previous work that searches in either mailing lists or issue-tracking systems for architectural knowledge, but there is little research done on the relationship between both. We know from exploratory studies that there are related pairs[13], but we don't have a good definition of their relations.

In open source software, a lack of documentation can hinder the use and future development of the open source software [6]. Despite its importance, Ding et al [6] found that 94.6% of OSS projects have no proper software architecture documentation. This could explain why developers often encounter difficulties in getting relevant architectural information for addressing quality concerns and making design decisions[4]. We attempt to solve this problem of finding relevant architectural information in open source systems. The goal of this paper is to *Explore the architectural relations between email threads and issue tracker issues that are about the same architectural design decision and provide architectural knowledge and how to find them using cosine similarity.* We are looking for cases where both an email thread and an issue are related to the same ADD and both provide architectural knowledge. We will define such an email thread issue pair as Architectural Design Decision Issue Email Pair (ADDIE-pair).

A pair of an issue and email thread that talk about the same ADD = ADDIE-pair

The rest of this paper has the following structure: Section 2 provides background on the theory, section 3 presents the research questions, used data and study design. Section 4 shows the results per research question and section 5 discusses these together with the implications for researchers practitioners and researchers. Section 6 talks about threats to validity and section 7 ends with the conclusion.

2 Background

2.1 Architectural Design Decisions

Architectural knowledge has been defined as the set of Architectural Design Decisions (ADDs). When we talk about ADDs we are using the ontology of Kruchten et al[9]. The three types of ADDs we use are 'Executive', 'Existence' and 'Property'. The following definitions are from the same paper:

2.1.1 Existence

Existence decisions ("Ontocrises") are about if something will exist or be present in the system. In the ontology of Kruchten et al there is also Non-existence decisions ("Anticrises") which talks about the absence or ban of something. They can bee seen as constraints in software development. We merge non-existence into existence since they both talk about whetere something needs to exist or not. It can be subdivided in structural and behavioral decisions, with structural talking about what exists and behavioral is about how elements interact.

2.1.2 Executive

Executive decisions ("Pereicrises") are decisions that are less about the software design or qualities but more about the business process around it, affecting how the development process takes place. They are often constrain existence and or property decisions. An example would be: "The code must we written in Java because our team only known Java." or "All public APIs must be approved by the board".

2.1.3 Property

Property decisions ("Diacrises") are about what properties the software must have. An example would be the requirement to use a specific version of software.

2.2 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them[7]. In order to calculate the cosine similarity between two vectors we can calculate their dot product. We can use cosine similarity to compare 2 texts, but we first need to vectorize our texts in order to calculate the dot product. We want to try two different vectorization approaches. One context based and one semantic based. For the context based approach we use Term Frequency Inverse Document Frequency (TF-IDF). For the semantic approach we use Sentence Bidirectional Encoder Representations from Transformers (SBERT).

$$\textbf{Cosine Similarity bwetween vector A and B} = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

2.2.1 TF-IDF Transformation

Term Frequency Inverse Document Frequency (TF-IDF) transform is a method to increase the significance of the 'uniqueness' of a word in its corpus[13]. It also reduces the impact of common words and increases the impact of distinctive words. TF-IDF is calculated by multiplying term frequency (TF) by inverse document frequency (IDF). Their formulas are:

$$\text{TF}(t, D) = \frac{\text{Number of occurrences of token } t \text{ in document } D}{\text{Total number of tokens in document } D}$$

$$\text{IDF}(t, D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right)$$

With t being a token, N being the number of documents in the corpus and D being the number of documents in the corpus in which the token t appears[13] .In this paper we used the TF-IDF transformation created by Pijnacker et al[13]. This is because the code was created for the data set we use, which is created by the same authors. This way we know the code is compatible with the data, and saves us time. An additional benefit is that they use stemming. This is a process of normalizing all variants of a word into one form (e.g. ‘creation’ and ‘create’ turn into ‘creat’).

2.2.2 SBERT

While TF-IDF is a context based variant for text vectorization, we also looked at a more semantic variant. An option could be using BERT, which is a natural language processing family based on the transformer architecture[5]. It can generate word embedding vectors from text, allowing us to calculate the dot product (cosine similarity) between two words. BERT has been pre-trained to know the meaning of words and their semantic relations. It achieved state of the art performance for the NLP tasks GLUE, SQuAD v1.1, SQuAD v2.0 and SWAG[5]. A downside with this method is that finding the most similar pair can take a long time, with a collection of 10,000 sentences requiring about 50 million inference computations (65 hours)[15]. A way to make this faster is to use Sentence-BERT (SBERT), a modification that use siamese and triplet network structures to derive semantically meaningful sentence embeddings. These sentence embeddings can be used as vectors for cosine similarity. This reduces the amount of time from 65 hours to about 5 seconds while maintaining the accuracy from BERT[15].

3 Study Design

3.1 Research Questions

To achieve our goal, we ask the following research questions:

– (**RQ1**) *How do software engineers discuss ADDs between issue tracker issues and email threads?* Finding how software engineers discuss the same architectural design decision in email threads and issues can help in capturing AK. We are looking for patterns that issue - email thread pairs follow and why.

– (**RQ2**) *What are the most effective similarity methods for finding issue - email thread pairs that discuss the same ADD?* Having an effective way to find these pairs is useful in capturing AK. We will look for issue - email thread pairs that are about the same ADD when at least 1 element of the pair (so either the issue or the email thread) is marked as containing a certain ADD type, whilst for the other element we do not have that requirement. This way we only need to mark one element of the pair to find its related pairs.

– (**RQ3**) *What are the characteristics of related issue - email thread pairs that discuss the same ADD?* If we could find different characteristics in the different patterns or type of pairs, then these characteristics could help in finding certain pairs more precise. It could also show us that certain pairs are better suited for different applications.

In order to answer those questions we need to first find the ADDIE-pairs. We will do a total of 5 iterations experimenting with different similarity methods to find ADDIE-pairs.

Iteration	Summary
0	Context based similarity (TF-IDF)
1	Context based similarity (TF-IDF) + Filter
2	Semantic based similarity (SBERT) + Filter
3	Model averaging of iteration 1 and 2
4	Increasing data in iteration 3

Table 1: Different iterations used to find ADDIE-pairs

At the end we will have created a new dataset containing email-issue pairs with a similarity value assigned to them. Of this dataset, a subset (at minimum 100 pairs in each table) with the highest similarity value will be manually judged if it is an ADDIE-pair or not. The data that we start with is explained in section 3.2. Based on these ADDIE-pairs we will apply further analysis in order to answer our research questions, which we will explore in section 4

3.2 Dataset

To answer these research questions we used the following data in a PostgreSQL database which was created by Pijnacker et al.[13] This database contained 70759 Jira issues and 43815 emails from the following six Apache projects: Cassandra, Hadoop, HDFS, MapReduce, Tajo and Yarn. Of these issues, 1426 were labeled as at least 1 type of ADD (according to the ontology of Kruchten et al.[9]). From the emails, 620 were labeled with at least 1 type of ADD (using the same ontology as with the issues).

During iteration 0 (see 3.3.2) we expanded the datasets in order to apply filtering. We also merged the ADD types into the email tables for better ease of use.
The email table schema was expanded with the following columns

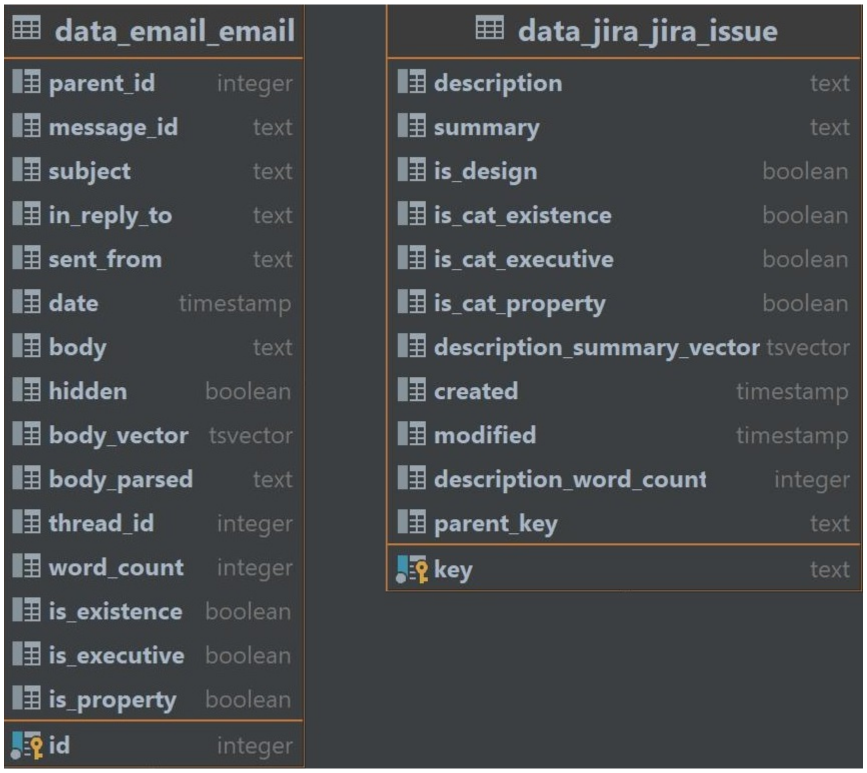
1. thread_id: The ID of the first email in the email thread (which can be its own ID).
2. word_count: The word count of the email body.
3. is_existence: Boolean that checks if the email is marked as an existence ADD
4. is_executive: Boolean that checks if the email is marked as an executive ADD
5. is_property: Boolean that checks if the email is marked as an property ADD

The ADD categories were taken from another table and are null-able since not every email has been labeled.

The Jira issues was expanded with the following two columns:

1. description_word_count: The word count of the description.
2. parent_key: The key of the parent Jira issue (which can be its own ID).

This results in the final schemas as shown in figure 1.



data_email_email	
parent_id	integer
message_id	text
subject	text
in_reply_to	text
sent_from	text
date	timestamp
body	text
hidden	boolean
body_vector	tsvector
body_parsed	text
thread_id	integer
word_count	integer
is_existence	boolean
is_executive	boolean
is_property	boolean
id	integer

data_jira_jira_issue	
description	text
summary	text
is_design	boolean
is_cat_existence	boolean
is_cat_executive	boolean
is_cat_property	boolean
description_summary_vector	tsvector
created	timestamp
modified	timestamp
description_word_count	integer
parent_key	text
key	text

Figure 1: Schemas of email table (left) and issue table (right)

3.3 Iteration 0: Context Similarity

3.3.1 Exploratory round

In iteration 0 we first did an exploratory round of open coding (according to grounded theory [18]). In order to find ADDIE-pairs we used cosine similarity to first find related email-issue pairs. The similarity values were calculated using the TF-IDF cosine similarity that we explained it section 2.2.1. Using this method we created 2 tables. The first table, named result_arch_emails_all_issues, was created by matching all architectural emails against all issues. The second table, named "result_arch_issues_all_emails", was created by matching all architectural issues against all emails. For both tables we only took the pairs that had a similarity score greater than 0.1 to save space. Both tables consist of the 3 columns: an ID to find the email (email_id, integer), an ID to find the issue (issue_key, string) and the similarity score (similarity, float between 0 and 1). We then used these tables a guide to find ADDIE-pairs. We took the first 25 pairs with highest similarity score from each table. Here we noticed in the result_arch_emails_all_issues table that 2 variables could help judge if a pair is a ADDIE-pair or not.

The first is the creation time difference, which is the amount of days between the creation of the email and the creation of the issue. Of the 8 non-ADDIE-pairs in the result_arch_emails_all_issues table, the creation time difference (in days, ascending order) were: 2, 119, 687, 999, 1486, 1664, 1859 and 2191. With the exception of 2 and 119, all creation time differences of ADDIE-pairs were smaller. An explanation for this could be that when an ADD is created or formulated in any form, be it email of issue, that discussion about it will be around the same time and not years into the future. It could be that an ADD of the past influences the creation of a future ADD, but those are not the same.

The second variable is the smallest word count. This is the smallest value of the email body word count and issue description word count. Out of the 8 non-ADDIE-pairs of the same table, the smallest word counts are (in ascending order): 0, 0, 3, 15, 24, 46, 61 and 76. 14 out of 17 ADDIE-pairs had a higher smallest word count. A reason for this could be that most architectural changes are big enough to require more than a few sentences to explain. According to van Tussenbroek, “TF-IDF’s accuracy increases linearly as the minimum word count set [...] changes”[19]. This could also explain why having a higher minimum word count increases the effectiveness of finding related pairs.

We also noticed in the result_arch_emails_all_issues table that there were pairs that link the same email thread to the same (parent) issue. If we have an email discussion it could be the case that multiple emails in that thread will get a high cosine similarity score with the same Jira issue. For example, if we look at the pair with issue key HADOOP-3246 [1] and the email with ID 2253[2] and compare it with the pair with issue key HADOOP-3246[3] and email with ID 2246[4] we see that both pairs are talking about the same ADD which gets implemented in the shared issue. Since we care about the entire path the ADD takes and the role emails and issues have in that path we will always look at the scope of the entire email thread and the issue with its children or parent. Since we are looking for unique relations of an ADD instead of unique pairs these duplicates clutters up our results.

After the exploratory round, we decided to enrich our data by adding the smallest word count, creation time difference, email thread id and issue parent key. We do this because we hypothesize that influencing those 4 variables could improve the precision of finding unique ADDIE-pairs.

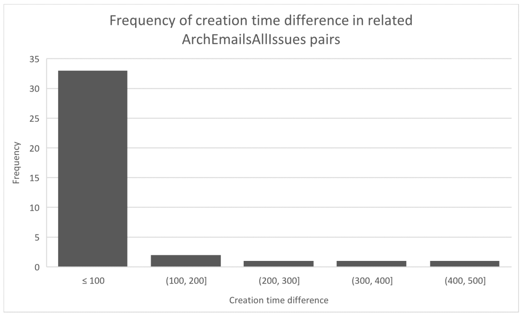
3.3.2 Open Coding Round

After enriching our data we did an open coding round where we took the first 100 pairs with the highest similarity score for both tables. For each pair we would label it either ’not related’ ’architectural irrelevant’ or a specific pattern that the conversation seemed to follow.
After the round was completed we got 8 patterns, but with the intent to refine them later once we got more data from future rounds and iterations. We also decided to make a histograms for creation time difference and smallest word count, to help us answer our hypothesis that smallest word count and creation time difference. The histograms can be seen in figures 2 and 3 The results from this round can be seen in table 2. We used this data to continue in iteration 1.

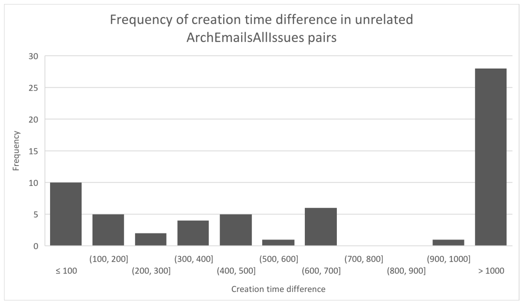
Iteration 0		
	ArchEmailsAllIssues	ArchIssuesAllEmails
Number of ADDIE-pairs	38	41
Number of unrelated pairs	62	59

Table 2: Results from Iteration 0: TF-IDF Cosine similarity

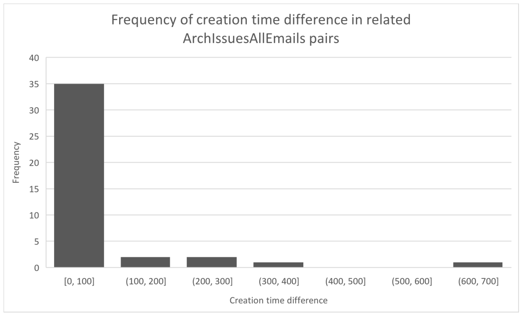
[1]Issue url = https://issues.apache.org/jira/browse/HADOOP-3246
[2]Email url = https://www.mail-archive.com/core-dev@hadoop.apache.org/msg07315.html
[3]Issue url = https://issues.apache.org/jira/browse/HADOOP-3246
[4]Email url = https://www.mail-archive.com/core-dev@hadoop.apache.org/msg07475.html



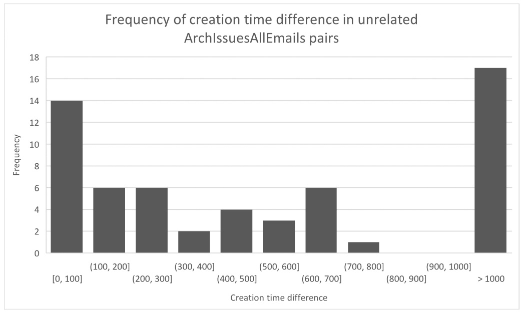
(a) Creation time difference histogram of ADDIE-pairs in ArchEmailsAllIssues



(b) Creation time difference histogram of unrelated pairs in ArchEmailsAllIssues

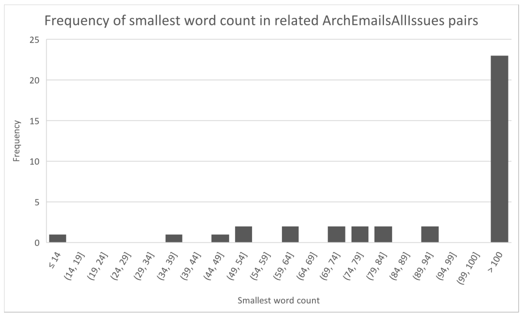


(c) Creation time difference histogram of ADDIE-pairs in ArchIssuesAllEmails

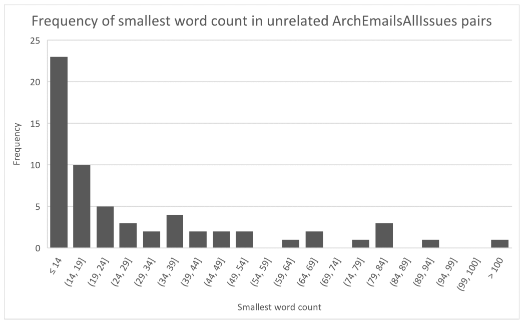


(d) Creation time difference histogram of unrelated ArchIssuesAllEmails pairs

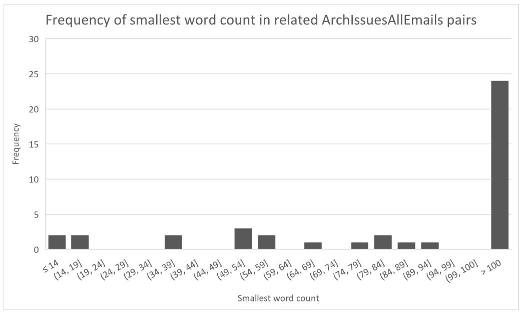
Figure 2: Creation time difference histograms



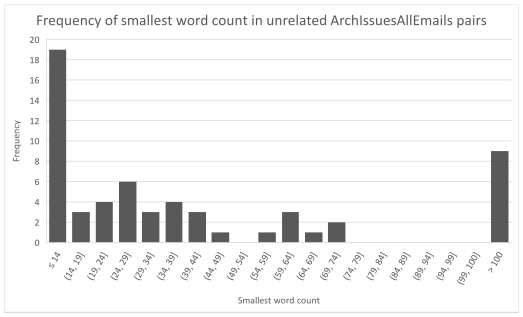
(a) Smallest word count histogram of ADDIE-pairs in ArchEmailsAllIssues



(b) Smallest word count histogram of unrelated pairs in ArchEmailsAllIssues



(c) Smallest word count histogram of ADDIE-pairs in ArchIssuesAllEmails



(d) Smallest word count histogram of unrelated pairs in ArchIssuesAllEmails

Figure 3: Smallest word count histograms

3.4 Iteration 1: Filtering

In iteration 1 we look at the results of iteration 0, to determine how we want to do our next open coding round. We looked at our hypothesis of iteration 0 that smallest word count, creation time difference, email thread id and issue parent key can influence the precision of finding ADDIE-pairs. We will first look at creation time difference.

3.4.1 Creation Time Difference

For determining the creation time difference we took the creation time of the email which is used in the comparison and not the creation time of the email thread. This is because somebody can respond to an email thread and create a new discussion which sparks a new ADD. By taking the creation time of each email instead of the email thread we can keep these new emails on old email threads. From this we got the histograms of figure 2, from which we can see that nearly all ADDIE-pairs fall in the bin of a creation time difference of 100 or less. Meanwhile in unrelated pairs the most common bin is a creation time of more than 1000, with the bin of 100 or less being second place. It could be the case that an ADD is created and that in the far future a new ADD is creation about the same subject or even build on top of the old ADD. If we want to find all email-issue pairs about a subject it could be better to get a wider creation time difference, but if we want to get only pairs about one ADD it is better to have a narrow creation time difference.
The question then arises: at what creation time difference do we stop? From the histograms of figure 2 we see two possible cut off values. One keeps all pairs with a creation time difference of 500 or less, and the other keeps all pairs with a creation time difference of 100 or less. If we take the first then we keep 98.73% of all ADDIE-pairs while keeping only 47.93% of unrelated pairs. If we take the latter then we keep 86.08% of all ADDIE-pairs whilst keeping only 19.83% of unrelated pairs. From this the cutoff at 100 seems to have a better accuracy, we but chose the cutoff at 500 in order to preserve more ADDIE-pairs. If we combine both tables we can see with a cutoff at 500, 98.44% above the cutoff are unrelated pairs. From pairs below the cutoff we see that 57.35% of pairs are ADDIE-pairs, giving us a nice increase in percentage of ADDIE-pairs compared to 39.5% without the filter. Since this is not the only filtering we will apply, having a higher percentage of ADDIE-pairs will be better since we are going to remove more unrelated pairs with other filters.

3.4.2 Smallest Word Count

Now we will take a look at the smallest word count. Here we see that a smallest word count greater than 100 is by far the most common in ADDIE-pairs, whilst in unrelated pairs the most common smallest word count is smaller or equal than 14. That the most common smallest word count is smaller or equal than 14 in architectural unrelated pairs could be explained by 2 reasons. The first is that the TF-IDF's accuracy increases when the minimum word count increases[19]. The other reason is that a very small text (less than 15 words) has less room to explain or formulate an ADD and probably is more about the same subject instead of an exact ADD. Since we are looking for pairs that are about the same ADD, pairs about the same subject is not enough. If we were interested in the same subject than it could work. We can confirm our hypothesis that smallest word count influences the precision of finding ADDIE-pairs, but what would be a suitable cutoff number to remove unrelated pairs but still finding most ADDIE-pairs? One option to remove many unrelated pairs is to set the cutoff at 100, resulting in only 8.26% of unrelated ones remaining. This has the downside of only keeping 59.49% of ADDIE-pairs. The cost of the increase in precision is too high for us because removing around 40% of ADDIE-pairs can influence our definition of the patterns. We therefore chose a cutoff at 50, resulting in us keeping 88.61% of ADDIE-pairs and 22.31% of unrelated pairs. If we combine both tables we can see that of pairs with a smallest word count above 50, 91.26% are unrelated, whilst of the pairs with a smallest word count of 50 or lower, 72.1% are ADDIE-pairs. These percentages in combination with us applying multiple filters gives us the confidence that a cutoff at 50 is suitable for better precision in finding ADDIE-pairs.

3.4.3 Duplicates

Finally we have the two variables email thread id and issue parent key. Since we want to find ADDIE-pairs that are about different ADDs, we hypothesized that those two variables could increase the precision of finding unique ADDIE-pairs. Since we will look at the entire email thread and the entire family of the Jira issue (so an issue and all its children or if it is a child we look at the child and its parent). We will call a pair a duplicate if and only if there already exists another pair in the same table with: the same email thread id, the same issue parent key and a higher similarity score. This means that the same pair can exists in both tables and not be deemed a duplicate. We see in table 3 that the majority of duplicates is in the ADDIE-pairs. Reason for this could be that an unrelated discussion the same subject as the ADD could be mentioned, whilst in a related discussion the chance is higher that the subject is mentioned more. This is however due to the very small sample size hard to say. Nevertheless we did notice that out of all 200 pairs we can remove 25 of them, which comes down to 12.5%. So by using email thread id and issue parent key to select only the pair with highest similarity we can increase the accuracy of finding unique ADDIE-pairs. An important note is that we need to do this as a last step since this filter is not commutative. We also did notice that there were other pairs that talked about the same ADD but we could not find a clear filter for them. They were either emails send twice, resulting that they are not part of the same email thread, or issues not marked as child but as related whilst functioning as a child issue.

Iteration 0		
	ArchEmailsAllIssues	ArchIssuesAllEmails
Number of ADDIE-pairs	38	41
Number of unrelated pairs	62	59
Number of duplicates in ADDIE-pairs	13	7
Number of duplicates in unrelated pairs	3	2

Table 3: Duplicate data of iteration 0

3.4.4 Filter

This all together affirms our hypothesis that smallest word count, creation time difference, email thread id and issue parent key influence the precision of finding unique ADDIE-pairs. In order to increase precision we apply all 3 filters together in order to get more results. The final filter looks like this:

1. Remove all pairs where the smallest_word_count is less than 50.
2. Remove all pairs with a creation_time_difference greater than 500 days
3. For all pairs with the same email_thread_id and issue_parent_key take the pair with the highest similarity and drop the rest.

with step 1 and 2 being interchangeable but step 3 needs to be executed as last. Steps 1 and 2 could also be applied by excluding those pairs before calculating the cosine similarity. Although this would speed up the overall calculation time by reducing the amount of calculations, we only though of this after already doing the calculations with filtering, resulting in us not using nor testing it.
We also tracked how much pairs we discarded by filtering. We can see from table 4 that after filtering we are only keeping 4.18% and 1.58% of all pairs. We did however notice that the similarity values were dropping very fast in the first 20 to 40, and afterwards stabilizing with a slower decline. This can be seen in appendix B. This could explain the very low percentage, as the similarity values seem to indicate that there are just not that many ADDIE-pairs. In order to check for theoretical saturation for the pattern assignment we did analyze 150 pairs of each table. We noticed that these extra 50 pairs each did not deliver any new results and most of them were unrelated.

Now we have a different way of collecting ADDIE-pairs we start our analysis of iteration 1, using the filter mentioned above. We did 1 round of open coding, analysing 100 pairs of each table with the highest cosine similarity score. The results can be seen in table 5. Here we see that we managed to find 20.25% more ADDIE-pairs and if we count duplicates in the previous iteration as unrelated pairs, it jumps to 61,02% more unique ADDIE-pairs in the same sample size compared to the previous iteration.

Iteration 1: Amount of pairs		
Filter	**ArchEmailsAllIssues**	**ArchIssuesAllEmails**
None	18059	133554
Word filter	6488	81898
Smallest word count- and creation time difference	2569	26613
Smallest word count-, creation time difference- and duplicate	754	2110

Table 4: Iteration 1 tables with pair count.

Iteration 1		
	ArchEmailsAllIssues	ArchIssuesAllEmails
Number of ADDIE-pairs	45	50
Number of unrelated pairs	55	50

Table 5: Results from Iteration 1: Filtering

3.5 Iteration 2: Semantic Similarity

In iteration 2 we did another round of open coding, this time we looked if a semantic based SBERT cosine similarity would provide different results. For an SBERT model, we looked at 2 options, ‘all-mpnet-base-v2’ and ‘all-MiniLM-L6-v2’. According to the Sentence Transformers documentation ”The all-mpnet-base-v2 model provides the best quality, while all-MiniLM-L6-v2 is 5 times faster and still offers good quality.”[14]. We deemed the quality good enough for our purpose and we thus chose the ‘all-MiniLM-L6-v2’ model for its better speed, enabling us to use large data sets without requiring super computers.
For generating the similarity values we took the following approach:

1. Make a dictionary containing email_id and email_body.
2. Make a dictionary containing issue_key and the concatenation of issue_description and issue_summary. We did this because there are cases where the summary contained more or different information than the description. Another reason was that the TF-IDF similarity also did this, so we will keep it for consistency.
3. For each body we calculate the sentence embedding using the ’all-MiniLM-L6-v2’ SBERT model.
4. We create a new dictionary of id as key and the embedding as value.
5. We performed cosine similarity on the embeddings of architectural email and all issue pairs and on the embeddings of architectural issue and all email pairs. We only kept pairs with a similarity value greater than 0.35 in order to save space.
6. We filtered the results using the same filter we created in iteration 1.

We once again tracked the amount of pairs we got before and during filtering, which can be seen in table 6. We see that we only keep 0.14% and 0.14% of all pairs, which is a lot lower than in iteration 1. This is however compensated by the far greater number of total pairs.

Iteration 2: Amount of pairs		
Filter	**ArchEmailsAllIssues**	**ArchIssuesAllEmails**
None	39179435	56137779
Word filter	1455709	2282536
Smallest word count- and creation time difference	279998	459888
Smallest word count-, creation time difference- and duplicate	55261	79239

Table 6: Iteration 2 tables with pair count.

We then began analysing the first 100 pairs with highest similarity again, with the results shown in table 7. These results seem comparable to iteration 0 and worse than iteration 1. This was not expected, since we thought that it would be an improvement of iteration 1. We did however notice that we found different ADDIE-pairs. 38.46% of ADDIE-pairs and 90,16% of unrelated pairs we found were new and did not appear in iteration 1. This means that, although not providing a greater precision, it can give us pairs that the TF-IDF cosine similarity was not able to give us.

Iteration 2		
	ArchEmailsAllIssues	ArchIssuesAllEmails
Number of ADDIE-pairs	36	42
Number of new ADDIE-pairs	13	17
Number of unrelated pairs	64	58
Number of new unrelated pairs	59	51

Table 7: Results from Iteration 2: SBERT

We did question why it was not as effective as iteration 1 but during the research we did not find a clear reason for the ineffectiveness. We did find out later that the SBERT all-MiniLM-L6-v2 model is not a good fit for large texts, as it truncate all texts after a maximum word limit of 256. There exist other models with a larger word limit, but those are often not higher than 512 if we want the same quality. We then checked iteration 1 and 2 for pairs where the highest word count is higher than 256 in order to see if this influences the results. We put the results in table 8, from which we saw that there is a similar proportion of architectural related pairs in both iteration 1 and iteration 2 with word counts greater than 256. This makes us conclude that the performance for both methods is similar for texts with a word count above 256 words.

	Iteration 1		**Iteration 2**	
	ArchEmailsAllIssues	ArchIssuesAllEmails	ArchEmailsAllIssues	ArchIssuesAllEmails
Total ADDIE-pairs	45	50	36	42
Word count > 256	38	25	27	27
Total unrelated pairs	55	50	64	58
Word count > 256	36	16	23	19

Table 8: Number of pairs where at least one element has a word count greater than 256 in iteration 1 and 2

3.6 Iteration 3: Model averaging

In iteration 1 and 2 were able to find different ADDIE-pairs. To increase precision we explored the option of model averaging of the two models. Averaging the similarity of the two methods would provide us both the semantic similarity of TF-IDF cosine similarity and the contextual similarity of SBERT. We applied this model averaging to both ArchEmailsAllIssues and ArchIssuesAllEmails tables, and the results are presented in table 9.

Iteration 3		
	ArchEmailsAllIssues	ArchIssuesAllEmails
Number of ADDIE-pairs	46	44
Number of unrelated pairs	54	56

Table 9: Results from Iteration 3: Model averaging

This approach resulting in slightly lower amount of ADDIE-pairs. However we did notice that the precision of iteration 1 and 3 were very close. While the top 20 pairs with the highest similarity had better precision in iteration 3, this advantage diminished later compared to iteration 1.

3.7 Iteration 4: Expanding data

We noticed that the architectural issues all emails pairs consistently outperformed the architectural emails all issues pairs. We hypothesized that this difference was because we have a architectural email dataset of only 620 emails from 128 different email threads. Meanwhile we have 1426 architectural issues. We wanted to test if increasing the architectural data set would increase precision. In order to do this we were able to find a bigger data set of the same issues but with more labeled, with a total of 2166 laballed as architectural compared to our previous 1426. We were however not able to find a better data set for architectural labelled emails. This meant that we could only test this hypothesis in the ArchIssuesAllEmails pairs. We then needed to decide which method we use to calculate our similarity values. We decided to use the model averaging of iteration 3 because we noticed that it is more precise in finding ADDIE-pairs in the first 20 pairs, but dropping afterwards. We expect that when we increase the data that better precision will continue for longer. The results are shown in table 10. We see a 34.78% increase in ADDIE-pairs when comparing iteration 4

	Iteration 3	Iteration 4
Number of ADDIE-pairs	46	62
Number of unrelated pairs	54	38

Table 10: Results from the first 100 pairs in Iteration 4 (Increasing architectural issue data) compared with iteration 3

with iteration 3, supporting our hypothesis that increasing the architectural data increases precision. We decided to analyze a total of 200 pairs, where we concluded that theoretical saturation was reached. This conclusion was reached based on the fact that we began to find very little ADDIE-pairs in the last 50 pairs. In total we found 82 ADDIE-pairs in the top 200 pairs, only finding 20 new ADDIE-pairs in the last 100. During this iteration we also refined our ADDIE-pair patterns, which we discuss in 4.1.

3.8 Resulting Dataset

After these 5 iterations we have a total of 26 tables, which can be found in appendix D. Of these tables 9 contain the data used to analyze each iteration, 2 contain the all analyzed pairs (with the difference being how the ADD types are determined), 6 contain the all similarity pairs with a cosine similarity greater than 0 (3 for SBERT and 3 for TF-IDF), 3 containing the email and jira data and 6 containing extended data which is used to prepare for later iterations. For future work the table unique_pairs could be the most beneficial since it contains 681 unique pairs judged if they are both about the same ADD and what ADD types the issue is.

4 Results

4.1 RQ1: Issue - email thread patterns about the same ADD

During each iteration we created patterns which ADDIE-pairs seem to follow and assigned them to each ADDIE-pair. We analyzed a total of 681 unique pairs, 163 of them were ADDIE-pairs, 432 were unrelated and 86 architecturally irrelevant. We assigned each ADDIE-pair a pattern during each iteration, refining the patterns after each iteration. We reached theoretical saturation in iteration 4. We got in total 6 patterns based on the total of 163 unique ADDIE-pairs. Table 11 shows how many times each pattern did occur.

4.1.1 Pattern 1: Initiate ADD in Email Thread and Discuss in Issue.

In this pattern there is a begin of the ADD in the email (thread), which is almost directly moved to an issue. The email (thread) contains either an ADD or a plan for an ADD. There is the option for a short architectural relevant discussion to take place in the email thread, but this is the exception rather than the rule. The initial email asks for feedback about its ADD proposal (e.g. ”What do you think?” or ”Is this something we need?”). Often there are simple reply emails that voice their approval of the proposal. An issue will be created shortly (in less than a week) afterwards, and often but not always a reply with an url

Pattern	Occurrences
1	19
2	33
3	26
4	25
5	49
6	11

Table 11: ADDIE-pair patterns and their occurrences

(or JIRA key) to an issue will be send. In this issue there exists architectural relevant information or an architectural discussion in the issue comments.

Example	email id = 34469 [5]	issue key = CASSANDRA-14448 [6]
Source	Rational of discussion	quote
email	Initializing ADD	"I'm working on some performance improvements of the lightweight transitions..." "...current CAS requires 4 round trips to finish ..." "I'm proposing the following improvements to reduce it to 2 round trips."
Email	Requesting action / feedback	"What do you think? Did I miss anything?"
Jira	Creating Jira and adding ADD in description	The Jira contains (nearly) the same text as the initial email.
email	Notifying about move to Jira	"Cool, create a jira for it, https://issues.apache.org/jira/browse/CASSANDRA-14448."
email	Discussing about (architectural) information in email thread	"... if we combine the prepare and quorum read to- gether... We can improve it by avoid executing the read, if the replica already promised a ballot great than the prepared one.
Jira	Discussing about (architectural) information in comments	Person responding to 'Combine prepare and quorum read together' with "It's a tradeoff though, not a clear cut optimization: it will certainly speed up the non- contended case, but every time the prepare fails, you will have wasted time/resources on some reads..."

Table 12: Example of pattern 1, with source being in chronological order

4.1.2 Pattern 2: ADD in issue with Email as Feedback Request

This pattern is almost entirely based in the issue. Only at one point in time there is an email sent out to get the recipients to take action, often in the request of feedback. It can be that an issue has been created but no activity has been shown. An email will be sent out with architectural relevant information (sometimes the description is copy pasted) in order to gain attention.

[5]email url = https://www.mail-archive.com/dev@cassandra.apache.org/msg12600.html
[6]issue url = https://issues.apache.org/jira/browse/CASSANDRA-14448

Example	email id = 14279 [7]	issue key = HADOOP-8803 [8]
Source	Rational of discussion	quote
Jira	Reason for creating ADD	"I am modifying the Hadoop's code ... to achieve better security."
Jira	Requesting action / feedback	"I want to know that whether community is interesting about my work? Is that a value work to contribute to production Hadoop?"
email	Notifying about existence of Jira	The same text as the Jira issue description, together with "I created JIRA for the discussion. https://issues.apache.org/jira/browse/HADOOP-8803#comment-13455025 "
Jira	(Optional) Discussing about (architectural) information in Jira issue comments	"Makes sense, but I think it's going to be difficult to plumb through the various abstractions here in a clean way that doesn't introduce specific dependencies on FileInputFormat, etc."

Table 13: Example of pattern 2, with source being in chronological order

4.1.3 Pattern 3: Create ADD in Email Thread and Implement in Issue

This pattern has the architectural design and discussion in the email thread and the implementation in the Jira issue. It often starts as a proposal or request for an ADD. Questions about the ADD are asked or answered and the ADD will be refined. Eventually a Jira issue will be created in order to begin working on the implementation of the ADD. There are also cases were an a person wants to submit an ADD to a project and sends an email about it. We include those emails also in this pattern because the issue is mostly about implementation and not about changing the ADD.

Example	email id = 8561 [9]	issue key = CASSANDRA-2017 [10]
Source	Rational of discussion	quote
Email	Initiating creation of ADD and requesting discussion	[Replacing ivy with maven-ant-tasks] "Is this something that people are OK with? It will result in the version details being specified from the build.xml and not a separate ivy.xml"
Email	Discussing about ADD	"Why? What are the advantages?" "1. It will make deploying to central easier ... 2. You seemed to think it would be better keeping all the version information in build.xml rather than in a separate file 3. ..."
Email	Requesting move to Jira / requesting start of implementation	"If everyone is OK I'll create a JIRA against Core with fixVersion 0.7 and add my patch there." NOTE: this was the first email but only after the last email in the thread was sent, was the JIRA issue created
Jira	Creating Jira containing ADD	"Replace ivy with maven-ant-tasks. Three main reasons: 1. In order to deploy cassandra to maven central, we will need to use maven-ant-tasks anyway 2. ... 3. ...
Jira	(Optional) Discussing about implementation of ADD in Jira issue comments	"Tested that ant release still works after realclean. On a standard build, dependency checking is dramatically faster than ivy's."

Table 14: Example of pattern 3, with source being in chronological order

[7] email url = https://www.mail-archive.com/common-dev@hadoop.apache.org/msg07358.html
[8] issue = https://issues.apache.org/jira/browse/HADOOP-8803
[9] email url = https://www.mail-archive.com/dev@cassandra.apache.org/msg01591.html
[10] issue = https://issues.apache.org/jira/browse/CASSANDRA-2017

4.1.4 Pattern 4: Release group

For pattern 4 we take a similar definition as W. Meijer's Release Group[12]. Here the email thread talks about the decision about releasing one or multiple issues (for example "symlink support in Hadoop 2 GA"[11]). Sometimes the reply emails can contain insightful discussion, sometimes a reply email consist of simply a vote (e.g. "+1"). Overall most if not all of the release discussion takes place inside the email thread, whereas the issue contains mostly information about an ADD. A difference between pattern 4 and the other patterns is that pattern 4 is less about the design of the ADD and is more focused on its release.

Example	email id = 28314 [12]	issue key = YARN-4356 [13]
Source	Rational of discussion	quote
Issue	Issue to be released	"*ensure the timeline service v.2 is disabled cleanly and has no impact when it's turned off*"
Email	Creating release discussion	"*I'd like to open a discussion on merging the Timeline Service v.2 feature to trunk (YARN-2928 and MAPREDUCE-6331) [1][2].*"
Email	Discussing about release	"*Big +1 on merging ATS-v2 to trunk. However, my concern to release it in 3.0.0-alpha (even as an alpha feature) is we haven't provide any security support in ATS v2 yet. Enabling this feature without understanding the risk here could be a disaster to end-user (even in a test cluster).*"
Email	Discussion reply	"*You're right. Can we document and clarify that it's still "alpha 1", and it doesn't have security features. I also think ATS 1.5 supports security features, so it's good for production - we should document it officially.*"
Email	"Conclusion"	"*Thanks everyone for chiming in on the discussion. Since no blockers were raised, I'll go ahead and start a vote thread.*"

Table 15: Example of pattern 4, with source being in chronological order

4.1.5 Pattern 5: Feature group

For pattern 5 we found that these pairs are very similar to the Feature Group defined by W. Meijer[12]. The ADDIE-pairs are less following a strict pattern and are more defined by other characteristics. Pattern 5 is assigned to ADDIE-pairs where the email thread discusses an ADD that contains other smaller ADDS. An example of such an email thread is the email titled "[DISCUSS] Hadoop SSO/Token Server Components"[14], where they list 8 required sub-components for their main component (that being the Hadoop SSO/Token Server Components). These sub-components can be seen as their own ADDS, resulting in this email being related to multiple ADDS. This could be seen as a collection of pattern 4s. This makes these email threads very big collections of architectural knowledge, but it is harder to link them to a specific issue, with the exception of umbrella issues. We assign this pattern to ADDIE-pairs if: 1) The email thread contains discussions about multiple ADDS; Or 2) The issue is a subtask of an issue which is related to an email thread like situation 1; Or 3) The issue is a collection of subtasks that are smaller ADDs; Or 4) The issue is an implementation of the ADD discussed in an email thread that discusses multiple ADD issues.
The issue of the pair from this pattern often contain little architectural discussion since the discussion already took place in the big email thread. Unlike other patterns which follow a pattern with their emails and issues, with pattern 5 we don't know anything about the chronological appearances of either the email thread or issue. Therefore the example in table 16 could differ from other cases. In our issue data set, we saw that (HADOOP) 'Project Rhino' and 'Cassandra Enhance Proposal (CEP)'[20] are most common in this pattern.

[11]email url = https://www.mail-archive.com/common-dev@hadoop.apache.org/msg10597.html
[12]email url = https://www.mail-archive.com/yarn-dev@hadoop.apache.org/msg23808.html
[13]issue url = https://issues.apache.org/jira/browse/YARN-4356
[14]email url = https://www.mail-archive.com/common-dev@hadoop.apache.org/msg09911.html

Example	email id = 17585 [15]	issue key = HADOOP-9392 [16]
Source	Rational of discussion	quote
Issue	Umbrella issue created	"*This is an umbrella entry for one of project Rhino's topic, for details of project Rhino, please refer to https://github.com/intel-hadoop/project-rhino/.*"
Email	Discussion email thread created	"*As a follow up to the discussions that were had during Hadoop Summit, I would like to introduce the discussion topic around the moving parts of a Hadoop SSO/Token Service. There are a couple of related Jira's that can be referenced and may or may not be updated as a result of this discuss thread.*"
Email	Listing of ADDS	"*Considering the above set of goals and high level interaction flow description, we can start to discuss the component inventory required to accomplish this vision: 1. SSO Server Instance:*" In total 8 componments are listen, which can be counted as ADDS
Email	Developing ADD	"*I have also updated our TokenAuth design in HADOOP-9392. The new revision incorporates feedback and suggestions in related discussion with the community, particularly from Microsoft and others attending the Security design lounge session at the Hadoop summit. Summary of the changes: 1. Revised the approach to now use two tokens, Identity Token plus Access Token, particularly considering our authorization framework and compatibility with HSSO;*" With more changes in the email.
Email	Discussion / conflict	Person is replying to a quote: "*"Personally, I think that continuing the separation of 9533 and 9392 will do this effort a disservice. There doesn't seem to be enough differences between the two to justify separate jiras anymore." Actually I see many key differences between 9392 and 9533. Andrew and Kai has also pointed out there are key differences when comparing 9392 and 9533. Please review the design doc we have uploaded to understand the differences.*"
Email	Discussion about sub-tasks	"*The following JIRA was filed to provide a token and basic authority implementation for this effort: https://issues.apache.org/jira/browse/HADOOP-9781*"

Table 16: Example of pattern 5, with source being in chronological order

4.1.6 Pattern 6: Initiate ADD in Issue and Discuss in Email Thread

This pattern is a mirrored version of pattern 1. Here an issue is created containing an initial ADD or idea, which is followed up by a discussion email thread. Often such a discussion would take place in the issue comments, but in order to get more attention an email is sent out. It therefore has a similar email intention to pattern 2, but where pattern 2 has the architectural relevant discussions in the issue comments, pattern 6 has them in the email thread. Optionally this pattern has an architectural discussion in the issue comments, but this is not always the case.

[15]email url = https://www.mail-archive.com/common-dev@hadoop.apache.org/msg09911.html
[16]issue url = https://issues.apache.org/jira/browse/HADOOP-9392

Example	email id = 18812 [17]	issue key = HDFS-5333[18]
Source	Rational of discussion	quote
Issue	Issue created containing ADD	"Issue description = *This is an umbrella jira for improving the current JSP-based HDFS Web UI."* and comment posted half an hour later by issue creator = "One task of this jira is to modernize the UIs, however, the most important task I want to achieve in this jira is to move from server-side JSP pages towards client-based, AJAX-styled HTML 5 web pages."
Email	Creation of email containing information about ADD	"*Jing Zhao and I recently have reimplemented the JSP-based web UIs in HTML 5 applications*" ... "*The abstractions between the UI and the core server are well-defined, decoupling the UI and the core hadoop servers.*"
Email	Requesting action / feedback	"*Your feedbacks are highly appreciated.*"
Email	Discussing about ADD	"*I have a few concerns about removing the old web UI, however: * If we're going to remove the old web UI, I think the new web UI has to have the same level of unit testing. We shouldn't go backwards in terms of unit testing.* "
Issue	(Optional) Discussion about ADD in issue comments	"I have concerns with this client-side js only approach, which is less secure than a progressively enhanced hybrid approach used by YARN. The recent gmail XSS fiasco highlights the issue."

Table 17: Example of pattern 6, with source being in chronological order

4.2 RQ2: Effective similarity methods of finding issue - email thread pairs that discuss the same ADD

Iteration	Summary
0	Context based similarity (TF-IDF)
1	Context based similarity (TF-IDF) + Filter
2	Semantic based similarity (SBERT) + Filter
3	Model averaging of iteration 1 and 2
4	Increasing data in iteration 3 (only ArchIssuesAllEmails)

Table 18: Definitions of the different iterations used

Iteration	ArchEmailAllIssue			ArchIssueAllEmail		
	unrelated	architecturally irrelevant	ADDIE	unrelated	architecturally irrelevant	ADDIE
0	45	17	38	43	16	41
1	39	16	45	34	16	50
2	61	3	56	50	8	42
3	46	8	46	43	12	45
4	–	–	–	23	15	62

Table 19: Results of the different iterations from the top 100 pairs in each table.

In order to answer the question of what similarity method would be best, we used 5 different ways of finding ADDIE-pairs with each method having a sample pool of 100 pairs. Figure 6 and 7 show the precision of each iteration, with figure 6 not having a graph for iteration 4 since we did not change anything for

[17] email url = https://www.mail-archive.com/hdfs-dev@hadoop.apache.org/msg11691.html
[18] issue = https://issues.apache.org/jira/browse/HDFS-5333

ArchEmailsAllIssues table during that iteration. We also put the numbers in table 19 where we split unrelated pairs into two categories. One category 'unrelated' contains the pairs that were not ADDIE-pairs but do contain architectural knowledge. The other category 'architecturally irrelevant' contains pairs where at least one element of the pair does not contain architectural knowledge. We see that the ArchIssuesAllEmails pairs outperforms the ArchEmailsAllIssues pairs. An explaination for this could be that we have more than twice as much architectural issues compared to emails. From these graphs we see that iteration 4 is clearly the most effective way of finding ADDIE-pairs. While we only apllied that method on the ArchIssuesAllEmails pairs we are confident that similar results could be obtainable when applied to ArchEmailsAllIssues.

We also saw in iteration 0 that with unfiltered TF-IDF cosine similarity, 91.26% of not related had an email or issue word count of less than 50. However if both email and issue have at least a word count of 50, then there is a 72.16% chance of the pair being an ADDIE-pair. Figure 4 shows us the frequency of the smallest word count on unrelated pairs and ADDIE-pairs, from which we can see that with a minimum word count of 50 (for both emails and issues) we can eliminate most unrelated pairs.

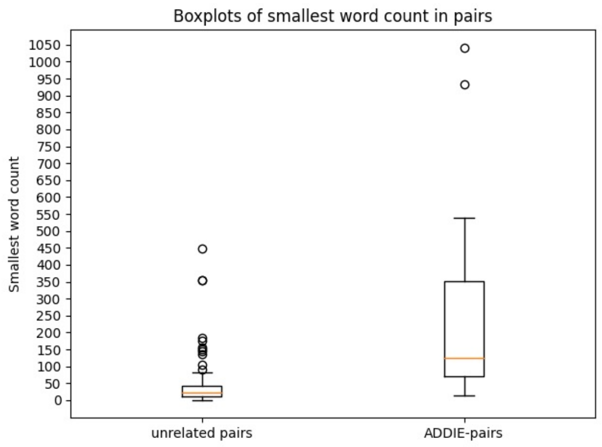


Figure 4: Boxplot of smallest word count in iteration 0 pairs, with ArchEmailsAllIssues and ArchIssuesAllEmails merged.

We also noticed that creation time difference played a big role for the effectiveness of similarity methods in finding issue - email thread pairs about the same ADD, with the ADDIE-pairs never having more than 700 days between the creation of the email and the issue. Here we take the creation time of each email and not just the creation time of the email thread. We determined that selecting pairs with a time difference of 500 or less between the creation of the email and the creation of the issue, that 57.35% of the pairs were ADDIE-pairs. Whilst we saw that if we took all pairs with a creation time difference greater than 500 that 98.44% of those pairs were unrelated. From this we conclude that with a maximum of 500 days between the creation of an email and issue we can eliminate most unrelated pairs.

For finding unique issue - email thread pairs that talk about the same ADD we a filter to reduce duplicates. For each email thread - parent issue combination, only taking the issue - email pair with the highest similarity score allowed us in iteration 0 to preemptively remove 12.5% of the pairs because they were duplicates. Although there are still duplicates, this filter will still increase the precision for finding unique issue - email thread pairs that talk about the same ADD.

When it came to comparing TF-IDF cosine similarity versus SBERT cosine we saw the following. When taking the top 100 pairs in both ArchEmailsAllIssues and ArchIssuesAllEmails tables and combining them, we saw that TF-IDF performed 21.79% better compared to SBERT. Despite having less precision, SBERT was able to find different ADDIE-pairs.
Taking the average of the two methods resulted in similar to precision as only using TF-IDF.

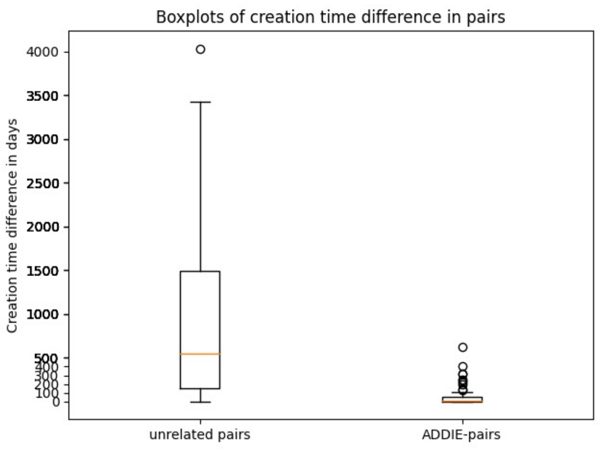


Figure 5: Boxplot of creation time difference in iteration 0 pairs, with ArchEmailsAllIssues and ArchIssuesAllEmails merged.

The amount of architectural labelled data also has influence on the similarity precision. We saw that the ArchEmailsAllIssues, with 620 architectural labelled emails, was outperformed by ArchIssuesAllEmails, with 1426 architectural labelled issues. This precision difference became even larger when we increase the architectural labelled issues to 2166 in iteration 4.

One notable occurrence that we found was that when the amount of ADDIE-pairs began to slow down, we did manage to find a lot of pair that were related on the same subject but not the same ADD. An example would be the issue HADOOP-1134[19] which says: "See recent improvement HADOOP-928 (that can add checksums to a given filesystem) regd more about it. Though this served us well there a few disadvantages" and ends by proposing a new ADD ("We propose to have CRCs maintained for all HDFS data in much the same way as in GFS."). If we look at the email with id 795[20] we see that they created the issue 'HADOOP-928'. The relation here is that the email with id 795 and issue 'HADOOP-928' are both about creating an ADD, whilst issue 'HADOOP-1134' then proposes a new ADD because of the other ADD. They are related on subject, but they are two different ADDs. We did not count this as an ADDIE-pair, but for future work this might be interresting to look into.

To summarize, our method that got us the best precision for ADDIE-pairs is to:

- Increase the architectural data set as much as possible.
- Remove all emails and issues with a word count smaller than 50.
- Calculate only similarity values for email issue pairs with a maximum creation time difference of 500.
- Calculate similarity values by taking the average of TF-IDF cosine similarity and SBERT embedding cosine similarity.

[19]issue url = https://issues.apache.org/jira/browse/HADOOP-1134
[20]email url = https://www.mail-archive.com/hadoop-dev@lucene.apache.org/msg07062.html

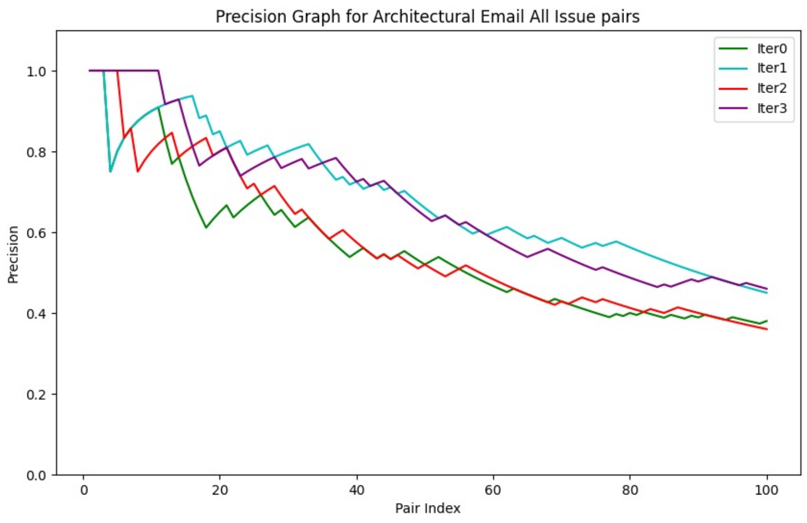


Figure 6: Precision graph of iterations 1 to 3 in Architectural Email All Issue pairs

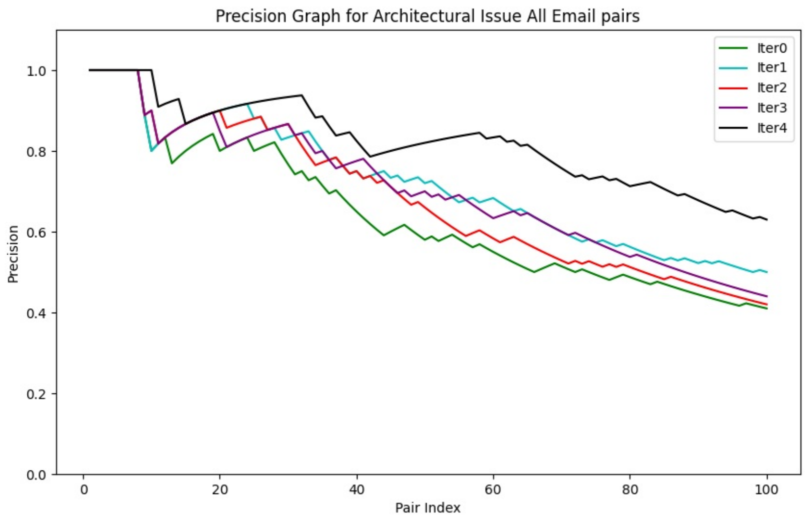


Figure 7: Precision graph of iterations 1 to 4 in Architectural Issue All Email pairs

4.3 RQ3: Characteristics of related issue - email thread pairs that discuss the same ADD

Now we have our 6 pattern, we can look if there are characteristics that set them apart from each other.

First we look at the amount of email per email thread. We know what patterns the ADDIE-pairs follow but we don't have a good estimate for how long certain discussion go on for. Looking at 8 we can see that pattern 5 has the most amount of emails per thread based on the IQR, but pattern 4 has the highest average. We can also see clearly that pattern 2 has a very small amount of emails per thread, which fits its pattern definition nicely. For pattern 4 we can justify the large amount of emails because it is about releasing and for every issue being released a discussion occur. Also there are a lot of replies which are simply about casting their vote (using for example "+1", as can be seen in email with id 32853[21]). As for pattern 5, since it is talking about very big and overarching ADDS or multiple ADDS, it expected that they have a lot of emails because there is a lot of things to discuss about.

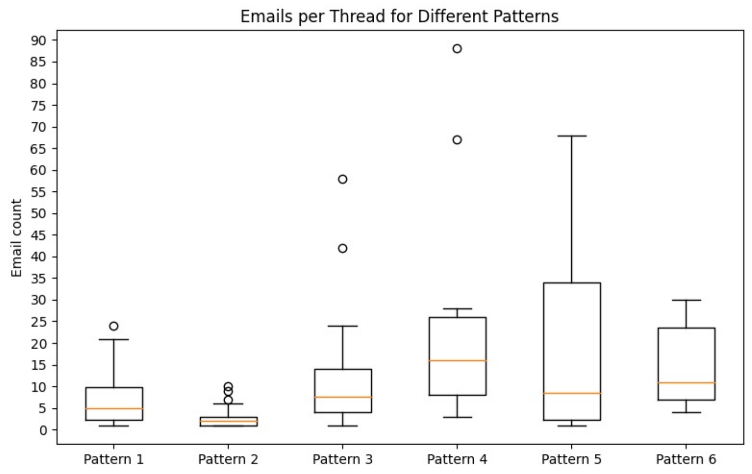


Figure 8: Boxplot of the amount of emails in a thread per pattern

The other place we can look is issue comments. From figure 9 we can see that patterns 2 and 5 have the opposite behavior as with emails in a thread. Pattern 2 has a lot of comments but few emails, and pattern 5 has many emails but few comments. For pattern 2 we can see that the main discussion takes place in the issue comments. Pattern 5 is also compliant with its pattern definition, since the discussion already took place in the big email threads there is little to discuss in the issue comments. It shows us two very different ways of developing an ADD.
The big range of pattern 4 can also be explained because the releasing process can be about any ADD, be it one that is created following pattern 2 or 5.

Looking at figures 8 and 9 we see that patterns 1 and 3 have around the same amount of emails in a thread and comments per issue. One difference of the two pairs is where the architectural discussion resides, but we can not see that from these boxplots.

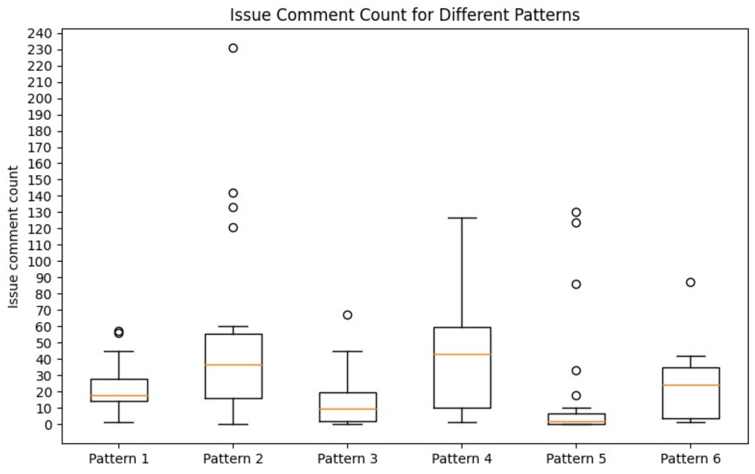


Figure 9: Boxplot of the amount of comments of an issue per pattern

[21]email url = https://www.mail-archive.com/mapreduce-dev@hadoop.apache.org/msg19032.html

We also looked if there was any relation between the pattern used to create the ADD and the ADD type. To test this we used the following Chi-square test hypotheses:

1. Is there a relation between the assigned patterns and the type of ADD?
2. Is there a relation between where the architectural discussion of an ADD takes place (issue or email thread) and the type of ADD?

To determine the type of ADD we have a few options. We can take the ADD type of either the email or the issue; We can take the union of the email and issue; We can take the intersection of the email and issue; Since we have at least one element of the pair labelled, there are cases where not both are labelled. This makes the intersection option fall off. Taking the ADD type from the email also has the downside that it could talk about multiple ADDs, resulting in the label being less accurate. With the union being less accurate but allowing us to use more pairs, we decided to use the following two ways:

A Take the ADD type from the issue (figure 10)

B Take the ADD type from the union (figure 11)

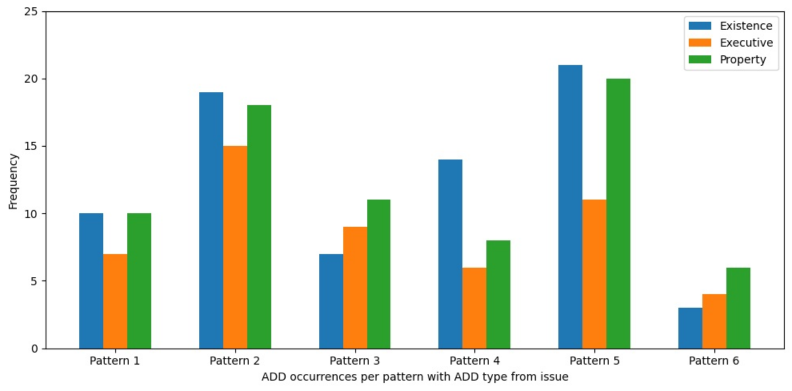


Figure 10: Histogram of the amount of each different ADD type per pattern, with ADD type taken from issue

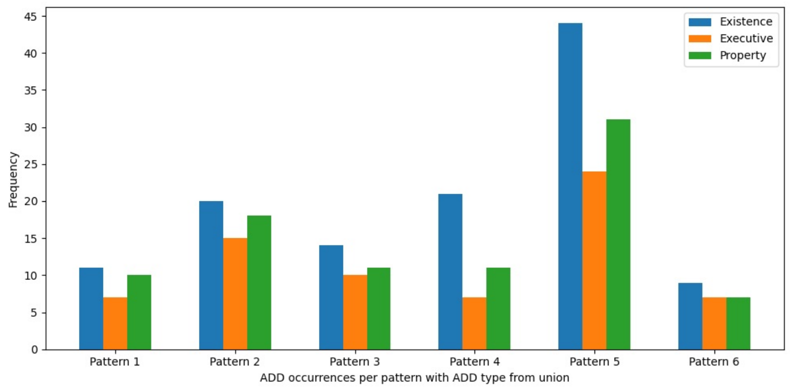


Figure 11: Histogram of the amount of each different ADD type per pattern, with ADD type taken from Union

For test number 2 we assigned all ADDIE-pairs of patterns 1 and 2 to the group where the architectural discussion takes place in the issue (comments), and assigned the ADDIE-pairs of patterns 3, 5 and 6 to the

	ADD type from issue (A)			ADD type from union (B)		
pattern	Existence	Executive	Property	Existence	Executive	Property
1	10	7	10	11	7	10
2	19	15	18	20	15	18
3	7	9	11	14	10	11
4	14	6	8	21	7	11
5	21	11	20	44	24	31
6	3	4	6	9	7	7
1 & 2	29	22	28	31	22	28
3 & 5 & 6	31	24	37	67	41	49

Table 20: ADD type per pattern

group where the architectural discussion takes place in the email thread. We excluded pattern 4 from the test since we can not know where the discussion takes place. The data we use can be seen in table 20.

Applying the Chi-squared test on these numbers gives us the results seen in table 21, with more detailled results found in appendix C. From this we can see that we have no p-value smaller than 0.05 meaning that the variables are independent and can reject our hypotheses.

Test hypothesis	Chi-square	degrees of freedom	p
1A	5.569702	10	0.850028
1B	3.551124	10	0.965334
2A	0.413865	2	0.8130746
2B	0.459911	2	0.7945690

Table 21: Chi-square test results

5 Discussion

5.1 RQ1: Issue - email thread patterns about the same ADD

Implications for practitioners The different patterns can help practitioners get insight in their developing process. It could guide practitioners in finding the related pair if they have only an email or issue, since the patterns show which options there exist.
Implications for researchers The patterns could function as a basis for future research on this topic. It could also be beneficial for researchers who want to get more insight how issues and email threads behave in relation to each other. We did for example notice at the end of iteration 4 that we were not finding a lot of pairs that discuss the same ADD, but we were able to find pairs that were talking about the same subject. Often we had two ADDs with one of them being an improvement or build upon the other.

5.2 RQ2: Effective similarity methods of finding issue - email thread pairs that discuss the same ADD

Implications for practitioners With the similarity method we tested in iteration 4 we showed how to find related pairs. This allows practitioners to run this on their corpus, which later then can be referred in order to find related pairs. This process needs to be redone in the future to include the newest pairs, but old pairs do not have to be recalculated. This allows practitioners to find related pairs more easily, allowing them to find architectural knowledge better.
Implications for researchers The similarity methods proposed in this study can provide as a way to find related pairs in different data sets. It also can function as a starting point for future research on more precise ways of finding related pairs. Certain variables such as author have not been taken into account, which could increase precision.

5.3 RQ3: Characteristics of related issue - email thread pairs that discuss the same ADD

Implications for practitioners The results produced from the chi-squared test do not add a lot of value of practitioners. The amount of comments on an issue can be an indicator for finding a related pair. If a practitioner finds an issue that contains an ADD but has very little comments then the practitioner can make a more narrow search.
Implications for researchers The characteristics allows researchers to get a better picture of the patterns. It could help in future research when looking for patterns. The chi-squared test shows that if we want to find certain ADD types such as property, that finding based on pattern is not effective.

6 Threats to Validity

6.0.1 External Validity

This work depends on data provided and used by other projects[13, 16, 12, 10]. The issues consist of open source project from Apache, which use Jira. This could threaten validity when the results are used for another issue tracking system or project. The same applies for the emails in the mailing lists, which are all from Apache projects.
Also the amount of labelled emails could be a threat with only a very small amount of the total being labelled. However we did see in section 3.7 that an increase in the amount of labelled issues increases performance, which explains could explain the difference.

6.1 Construct Validity

The low amount of occurrences of pattern 8 could be a threat. We did analyze 900 pairs, with 681 of them being unique issue-email pairs. Here we did reach theoretical saturation and thus covered most type of pairs between email theads and issues that talk about the same ADD.

We did also notice that our data contained a broken email thread sometimes and here the email thread with the title '[DISCUSS] Security Efforts and Branching'[22] was broken and caused a thread which should contain 3 emails to appear as a thread containing only 1 email. So instead of getting 1 thread containing 3 emails we get 3 entries containing 1 email. This influences the amount of pairs and the amount of emails in a thread.

6.2 Reliability

In order to make sure the pattern assignment went well, I checked every week with my supervisor on assigned patterns that were difficult to judge and we went over all patterns to refine them. This helps the reliability since the patterns and assignments have been validated and evaluated. There still is a risk since my supervisor and me have the same western cultural background and are from the same university which might not be optimal since we now have no cross cultural validation.

7 Conclusion and Future Work

In this paper we explored issues and email threads that talk about the same ADD. Using grounded theory we analizied 681 unique pairs, of which 163 were deemed related. We got these pairs using TF-IDF cosine similarity and SBERT to vectorize the texts and then applied cosine similarity. We also tested the effect model averaging, increase the data and applying a filter had to finding related pairs. We found that these pairs can follow 6 different patterns, 4 of which are based on chronological order of the discussion and 2 based on the purpose and context. We also looked at some characteristics of these pairs, showing us how they differ from each other. The results and data produced by this study allows industry to better find architectural

[22]email url = https://www.mail-archive.com/common-dev@hadoop.apache.org/msg10524.html

knowledge, and provides a starting point for future research.

In terms of future work we think that the following could be useful: Further research in the effects of increasing the size of architectural labelled data and research in finding pairs that are successor or predecessor ADDs. About the former we noticed that increasing the amount of architectural labelled data increased precision but that increase disappeared sooner than we hoped. An indicator could be the trend or rate of decline of the similarity values. We noticed that once the trend stabilized that the amount of related pairs that talk about the same architectural design decision tended to drop. We did however not look further into this. Another thing we noticed was that in the unrelated pairs we found that there are pairs where the ADD of one element is a successors or predecessor of the other element of the pair. We only recognized this trend in a later stadium so we did not keep exact track of them, but we noticed from out notes that at least 70 out of 432 unrelated pairs are these successor or predecessor pairs. Further research into this could help find related ADDs.

References

[1] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. Automatic extraction of design decisions from issue management systems: A machine learning based approach. In Antónia Lopes and Rogério de Lemos, editors, *Software Architecture*, pages 138–154, Cham, 2017. Springer International Publishing.

[2] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, 116:191–205, 2016.

[3] Rafael Capilla, Francisco Nava, and Juan C. Duenas. Modeling and documenting the evolution of architectural design decisions. In *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*, pages 9–9, 2007.

[4] Musengamana Jean de Dieu, Peng Liang, and Mojtaba Shahin. How do developers search for architectural information? an industrial survey, 2021.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[6] Wei Ding, Peng Liang, Antony Tang, Hans Vliet, and Mojtaba Shahin. How do open source communities document software architecture: An exploratory survey. 08 2014.

[7] Wael Gomaa and Aly Fahmy. A survey of text similarity approaches. *international journal of Computer Applications*, 68, 04 2013.

[8] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120, 2005.

[9] Philippe Kruchten, Patricia Lago, and Hans van Vliet. Building up and reasoning about architectural knowledge. In Christine Hofmeister, Ivica Crnkovic, and Ralf Reussner, editors, *Quality of Software Architectures*, pages 43–58, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[10] Andrew Lalis. Searching for architectural design decisions in open-source software mailing lists. Bachelor's thesis, University of Groningen, 2022.

[11] Umme Ayda Mannan, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. On the relationship between design discussions and design quality: A case study of apache projects. New York, NY, USA, 2020. Association for Computing Machinery.

[12] W. Meijer. Exploring the rationale of design decisions in open-source software mailing lists and their relationship to architectural issues. Master's thesis, University of Groningen, 2022.

[13] Bjorn Pijnacker, Jesper van der Zwaag, and Julian Pasveer. Exploring the cosine similarity for automated relating of architectural issues and emails in mailing lists. Evidence-based software engineering course project, master programm, University of Groningen, 2023.

[14] Nils Reimers. Pretrained models - sentence-transformers documentation. https://www.sbert.net/docs/pretrained_models.html. Accessed: 14-06-2023.

[15] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[16] Mohamed Soliman, Matthias Galster, and Paris Avgeriou. An exploratory study on architectural knowledge in issue tracking systems, 2021.

[17] Mohamed Soliman, Matthias Riebisch, and Uwe Zdun. Enriching architecture knowledge with technology design decisions. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 135–144, 2015.

[18] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: A critical review and guidelines. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 120–131, 2016.

[19] Thomas van Tussenbroek. Who said that? comparing performance of tf-idf and fasttext to identify authorship of short sentences. Bachelor's thesis, Delft University of Technology, 2020.

[20] Michael Semb Wever. https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=95652201, Jul 2023. Accessed: 14-06-2023.

A Code

The code for calculating the cosine similarities can be found here:
https://github.com/MKruijer/PythonSimilarityCode
The code for analyzing those results and applying the filter can be found here:
https://github.com/MKruijer/databaseEditor

B Similarity values per iteration

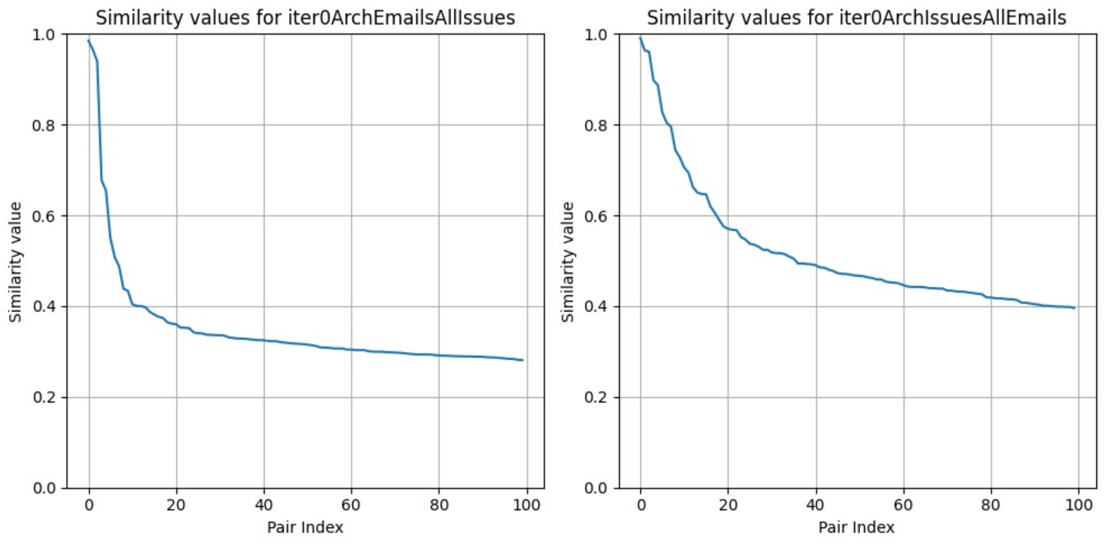


Figure 12: Graph of similarity values from iteration 0

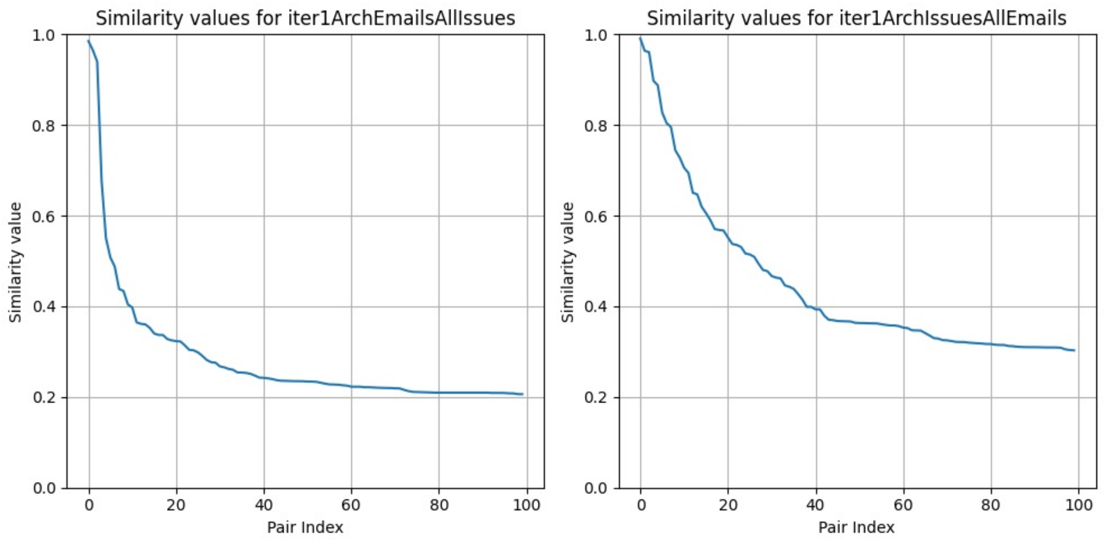


Figure 13: Graph of similarity values from iteration 1

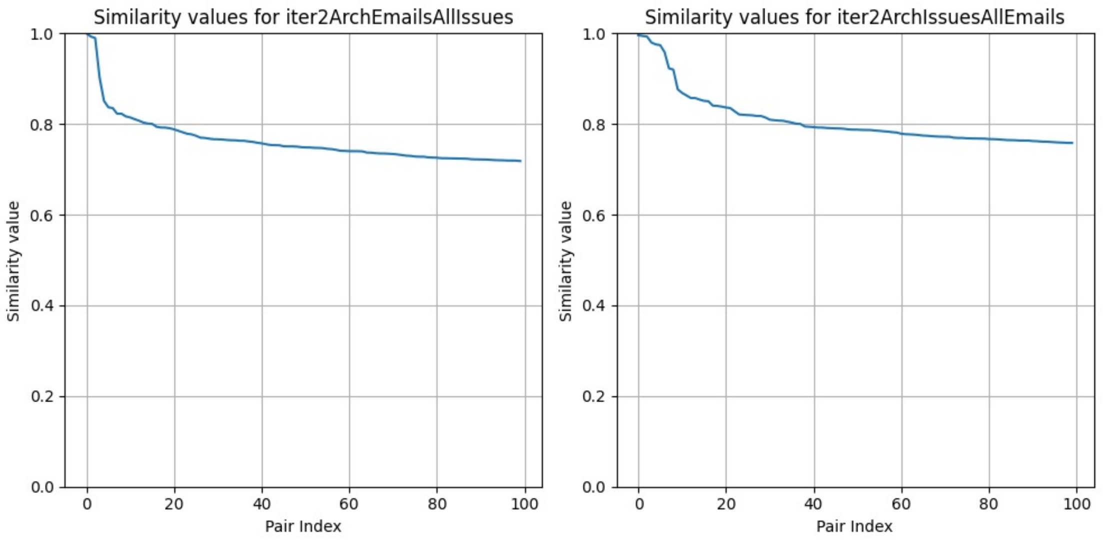


Figure 14: Graph of similarity values from iteration 2

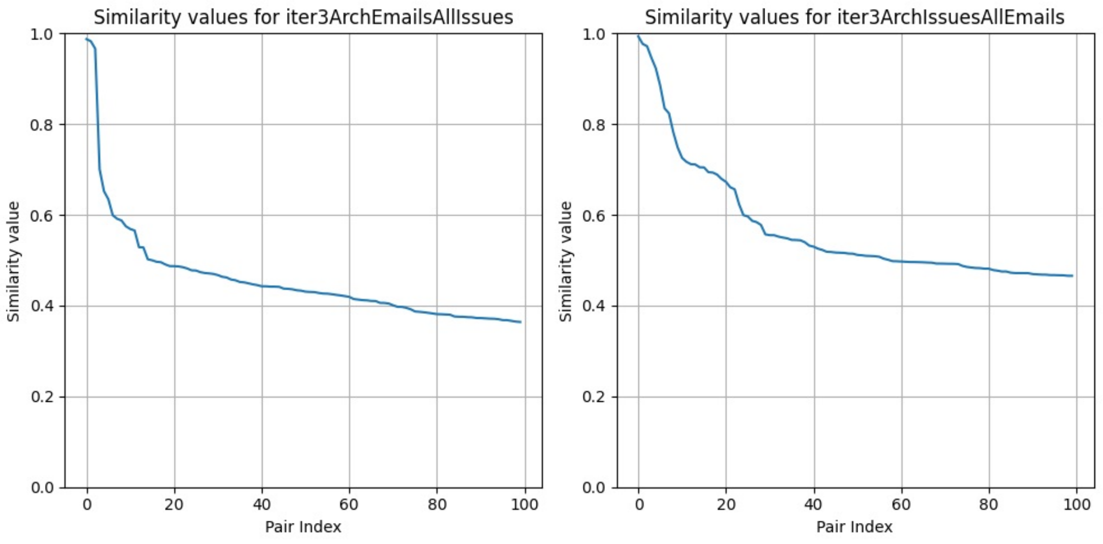


Figure 15: Graph of similarity values from iteration 3

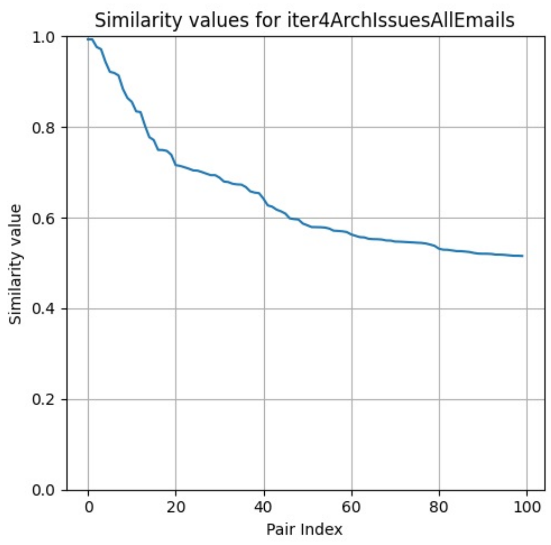


Figure 16: Graph of similarity values from iteration 4

C Chi-Square Data

	ADD type from issue (A)			ADD type from union (B)		
pattern	Existence	Executive	Property	Existence	Executive	Property
1	10.04	7.06	9.90	12.03	7.08	8.90
2	19.33	13.59	19.08	22.77	13.39	16.84
3	10.04	7.06	9.90	15.04	8.84	11.12
4	10.41	7.32	10.27	16.75	9.86	12.39
5	19.33	13.59	19.07	42.53	25.02	31.45
6	4.83	3.40	4.77	9.88	5.81	7.31
1 & 2	27.72	21.25	30.03	33.35	21.44	26.21
3 & 5 & 6	32.28	24.75	34.97	64.65	41.56	50.79

Table 22: Chi-squared expected ADD type per pattern

	ADD type from issue (A)			ADD type from union (B)		
pattern	Existence	Executive	Property	Existence	Executive	Property
1	0.00	0.00	0.00	0.09	0.00	0.14
2	0.01	0.15	0.06	0.34	0.19	0.08
3	0.92	0.54	0.12	0.07	0.15	0.00
4	1.24	0.24	0.50	1.08	0.83	0.16
5	0.14	0.49	0.04	0.05	0.04	0.01
6	0.70	0.11	0.32	0.08	0.24	0.01
1 & 2	0.06	0.03	0.14	0.17	0.01	0.12
3 & 5 & 6	0.05	0.02	0.12	0.09	0.01	0.06

Table 23: chi squared values for ADD type per pattern

D Database tables

- **data_email_email**: Contains the data of all emails.
- **data_jira_jira_issue**: Contains the data of all issues, with the exception of the comments.
- **data_jira_jira_issue_comment**: Every comment from each issue.
- **iter0_expanded_arch_emails_all_issues**: The ArchEmailsAllIssue TF-IDF pairs with expanded data. Takes all pairs from result_arch_emails_all_issues with a similarity greater than 0.1. This table is used in iteration 0.
- **iter0_expanded_arch_issues_all_emails**: The ArchIssuesAllEmail TF-IDF pairs with expanded data. Takes all pairs from result_arch_issues_all_emails with a similarity greater than 0.1. This table is used in iteration 0.
- **iter1_filtered_arch_emails_all_issues**: The ArchEmails results from iteration 0 with the filter applied. This table is used in iteration 1.
- **iter1_filtered_arch_issues_all_emails**: The ArchIssues results from iteration 0 with the filter applied. This table is used in iteration 1.
- **iter2_expanded_arch_emails_all_issues**: The ArchEmailsAllIssue SBERT pairs with expanded data. Takes all pairs from sim_result_arch_emails_all_issues with a similarity greater than 0.35. This table is used to prepare iteration 2.
- **iter2_expanded_arch_issues_all_emails**: The ArchIssuesAllEmail SBERT pairs with expanded data. Takes all pairs from sim_result_arch_issues_all_emails with a similarity greater than 0.35. This table is used to prepare iteration 2.
- **iter2_filtered_arch_emails_all_issues**: Applying the filter to the iter2_expanded_arch_emails_all_issues table and using it for iteration 2.
- **iter2_filtered_arch_issues_all_emails**: Applying the filter to the iter2_expanded_arch_issues_all_emails table and using it for iteration 2.
- **iter3_average_similarity_arch_emails_all_issues**: Contains the results from taking the average similarity of ArchEmailsAllIssues from iteration 1 and 2 and using it in iteration 3 (Model averaging).
- **iter3_average_similarity_arch_issues_all_emails**: Contains the results from taking the average similarity of ArchIssuesAllEmails from iteration 1 and 2 and using it in iteration 3 (Model averaging).
- **iter4_average_similarity_arch_issues_all_emails**: The average similarity of the updated SBERT and TF-IDF similarity values for ArchIssues. It takes them from iter4_cos_sim_filtered_arch_issues_all_emails (TF-IDF) and iter4_sen_sim_filtered_arch_issues_all_emails (SBERT) and is used in iteration 4.
- **iter4_cos_sim_expanded_arch_issues_all_emails**: Takes the pairs from iter4_cos_sim_result_arch_issues_all_emails with a similarity greater than 0.1 and expands the data. It is used to prepare iteration 4.
- **iter4_cos_sim_filtered_arch_issues_all_emails**: Applies the filter to iter4_cos_sim_expanded_arch_issues_all_emails. This table is used to prepare iteration 4.
- **iter4_cos_sim_result_arch_issues_all_emails**: The updated TF-IDF cosine similarity values of all ArchIssuesAllEmails pairs with a score greater than 0. These values are calculated using the updated / increased architectural labelled data. It is used to prepare iteration 4.
- **iter4_sen_sim_expanded_arch_issues_all_emails**: Takes the pairs from iter4_sen_sim_result_arch_issues_all_emails with a similarity greater than 0.1 and expands the data. It is used to prepare iteration 4.
- **iter4_sen_sim_filtered_arch_issues_all_emails**: Applies the filter to iter4_sen_sim_expanded_arch_issues_all_emails. This table is used to prepare iteration 4.

- **iter4_sen_sim_result_arch_issues_all_emails**: The updated SBERT cosine similarity values of all ArchIssuesAllEmails pairs with a score greater than 0. These values are calculated using the updated / increased architectural labelled data. It is used to prepare iteration 4.
- **result_arch_emails_all_issues**: The original TF-IDF cosine similarity values of all ArchEmailsAllIssues pairs with a score greater than 0. It is used to prepare iteration 0.
- **result_arch_issues_all_emails**: The original TF-IDF cosine similarity values of all ArchIssuesAllEmails pairs with a score greater than 0. It is used to prepare iteration 0.
- **sim_result_arch_emails_all_issues**: The original SBERT cosine similarity values of all ArchEmailsAllIssues pairs with a score greater than 0. It is used to prepare iteration 0.
- **sim_result_arch_issues_all_emails**: The original SBERT cosine similarity values of all ArchIssuesAllEmails pairs with a score greater than 0. It is used to prepare iteration 0.
- **unique_pairs**: All the unique email thread - issue pairs that have been judged as either ADDIE-pair, unrelated or architecturally irrelevant. The ADD type is taken from only the issue.
- **unique_pairs_union_add**: The same as unique_pairs but with the ADD type being from the union of the issue and email instead of just the issue.