university of groningen / faculty of science and engineering / bernoulli institute

Exploring architectural knowledge in issue tracking systems

Bachelor's Project Thesis

Twan Hoven, s3747298
Supervisor: Dr. M.A.M. Soliman

Abstract: Every software project, even on a small scale, has a software architecture. Some architectures may be well developed and carefully designed while others are created on the fly as seemed fit. We propose to explore existing software systems to capture architectural knowledge based on quantitative text-analysis of issues found in issue tracking systems. The goal is to be able to capture architectural knowledge that would have been unavailable otherwise.

1 Acknowledgments

Dr. M.A.M. Soliman was of great help during the study and I would like to thank him for that. The weekly meetings and guidance during the research contributed to the knowledge that has been captured within this paper.

Contents

2 Introduction

Software architecture consists of a series of design decisions that make up the software structures [1]. The architecture determines how these structures interact with each other and what their purpose is. Good architectures can have a big positive impact on the life cycle of a system in terms of modifiability, re-usability and overall quality [1]. This is why software architecture is a very important subject within the field of Computing Science and software engineering. It is difficult to learn, and takes a lot of experience to get right. It has to be mentioned that a system that adheres to architecture is only part of the equation. An architecture that is not documented is only usable as long as the developer or team of developers is still actively working on the project. As stated by Kruchten the best architecture consists of design decisions that are explicit and documented. An explicit decision is a decision that is made for a very specific reason [2].

In the real world, however, these design decisions are often not well, or not at all, documented. Part of the problem is that architectural design decisions are difficult to take and document. As of today, there is no consensus on how to define or formalize design decisions. One starts with the decision itself but after that different approaches have been proposed. As described by Jansen and Bosch, rules, constraints and requirements are the main essential elements [3]. This does not mean that the decisions have to be lost forever. Generally, the issues in issue tracking systems contain a lot of information on what has changed in the system over time. We propose to capture architectural knowledge by looking at the commits, a set of changes to the source code, and the issues, a textual description of the changes, that go with them. Exploring the information that can be extracted from the code and the issues results in a list of annotations that describe the high-level architectural knowledge and the coding book that belongs to it. Because documentation is often lacking, it is all the more important that this knowledge is recorded after the fact.

Previous studies on the subject have tried to capture the design decisions using various techniques. The ADDRA approach is one of the current approaches to capture architectural knowledge. ADDRA tries to recover architectural design decisions based on code from snapshots of a particular system [3]. However, the research only looks at the code which is only half of the relevant information. When exploring architectural knowledge we propose to also include the issue tracking system that is used during the development and maintenance to explore the knowledge that is hidden in the issues.

An example of research that does look into issue tracking systems is the research by Shahbazian et al [4]. It introduces architecture recovery using RecovAr, where code changes and issues are combined to create a decision graph [4]. Contrary to what we plan to do, their research did not involve textual analysis of the issues. The reason we propose to perform the textual analysis is that the issues contain a lot of unstructured information and knowledge that when annotated correctly can give an insight into the design decisions behind the code changes.

In a study by Shanin et al [5] AK concepts were captured using a conceptual model. This model is relatively simple compared to models used in similar research and focuses more on capturing higher level specifics in software documentation. The model tries to capture differences between requirements, concerns, alternatives or decisions and the relations between those concepts [5]. Although Shanin concluded that the AK conceptual model was able to help junior architects to recover architectural knowledge [5], the knowledge that is captured does not describe AK in much detail. This paper aims to capture AK in more detail by including new AK concepts described by Soliman et al [6].

The downside of the aforementioned approaches is that they are limited in terms of describing the design decisions. This study proposes to explore and analyze large existing open-source software systems using reverse engineering and textual analysis of issues found in issue tracking systems used during the development of such projects. The main goal of this study is to capture the present knowledge about the architecture, hidden in the issues. The knowledge captured would contribute to a better recognition of the available knowledge that up til now could not been captured.

3 Background information

Issue tracking systems
An Issue Tracking System, or ITS, to keep track of the state of a system. Issue tracking systems include way more information regarding the state of the project at hand than the name suggests. These systems are actively used within organizations that have in-house development teams, regardless of the size of these teams.

The core of this research involves looking at the issues that are maintained within these issue tracking systems. Each issue has a description that states the purpose and has a comment section in which developers can discuss the proposed idea and contribute to it. The purpose of the issue can range from bugs that need to be addressed to entirely new features because of changing client requirements. The issues discussed in this paper are mainly focused on adding new functionality as both of the projects at hand are already several.

Another benefit of an issue tracking system is that it is tightly coupled to the source code. Whenever someone works on an issue and updates code or data related to that issue this will be coupled. For this research, however, we will not consider the code changes that belong to every issue.

Architectural knowledge
Architectural knowledge and the design decisions that are part of this can be quite difficult to document. According to a study by Jansen Bosch in 2008 the decisions should at least contain the most essential parts such as rules, constraints and requirements [3]. However, according to Kruchten, relationships are also an important element such that all decisions can be structured [2]. The aforementioned elements are all based on qualitative rationale. This requires skill from the architect who needs to be able to communicate why a decision is made to, for example, stakeholders [1]. As proposed by Tang and van Vliet there can also be quantitative elements such as cost or risk assessments[7]. In order to gain an insight in such a complex system of knowledge a coding book is used as a means to classify or label decisions and knowledge. As part of a similar study by Soliman et al. [6] a coding book was created for the same purpose.

The coding book describes all the AK concepts that will be considered in detail. *Architectural Design Configuration (CONF)* describes the relationships between components of the proposed solution.

The *Architectural Component Behaviour And Structure (CB)* describes the behaviour and structure of singular components as well as the technical details or input and output data.

Solution Benefits and Drawbacks (BD) indicate the reasons why a certain solution might or might not be usable. The *Existing System Architecture Description (EX)* is a similar concept to the CONF and CB except that they apply to the already existing system instead of to the newly proposed solution.

Next to the previously mentioned concepts the following concepts will be considered; *Motivation of Design Issue (MOT)*, *User Requirements (UREQ)*, *Quality Attribute Requirements (REQ)*, *Technical Debt (TD)*, *Run-time Quality Issues (EXQ)*, *Contextual Constraints (CC)*, *Other System Architectural Solutions (OSAS)*, *Architectural Tactics (AT)*, *Assumptions (AS)*, *Trade-offs (TO)* and *Risks (R)*.

4 Research Process

In order to explore what architectural knowledge is present in issue tracking systems, we first need to know what types of architectural knowledge concepts are present. This leads to the following research question:

Research Question 1: *"What kind of architectural knowledge is present in issue tracking systems?"*

As an extension of this, we look at the location of where the different concepts are discussed as there may be a difference between the introduction, summary and comment sections of an issue. This results in the research question:

Research Question 2: *"In which issue sections are AK concepts for existence ADDs being discussed?"*

After establishing a broader overview of the concepts within the issues we will take a closer look at individual sets of knowledge. During the feasibility study it became clear that within the comments of an issue, multiple different concepts could be found. In order to get an insight in what different types of concepts co-occur we will answer the following question:

Research Question 3: *"How do AK concepts co-occur with each other to make ADDs in issue tracking systems?"*

Not only the co-occurrence of different concepts can be interesting but also the textual analysis and wording used within these comments on the AK, leading to the following questions:

Research Question 4: *"What textual variants are used when describing AK concepts in issue tracking systems?"*

and:

Research Question 5: *"What does the taxonomy of AK concepts look like in issue tracking systems?"*

Answering these research questions should give insight in the architectural knowledge concepts that are relevant and showcase their specifics and relations that help identify them. To answer our research question, we will follow three steps which form the research process.

1. **Select Issues**
   The most important step is to select the projects that serve as a starting point. Based on earlier research done by Soliman et al [6] three projects were selected that could provide enough issues with the necessary amount of content needed. For this research the Hadoop project [8] was selected as no research was done on that project thus far.

   As Hadoop is the overarching project this would be considered too large and too complex to serve as a base for this research. However, as Hadoop is build up using multiple smaller projects, these smaller projects are considered to be less complex and more useful in terms of data gathering. The selected projects are Hadoop-Common (Hadoop) and the Hadoop Distributed File System (HDFS). A total of 121 issues were selected following the process described by Soliman et al [6].

2. **Annotating the selected issues**
   During the annotation process every issue is then annotated using the concepts described in the coding book [6] while also maintaining a separation between the different subsections in each issue such as the description or comment section in order to be able to analyze these different sections later on.

   As described by Soliman et al. a set of decision factors exists to annotate the selected issues. One of the decision factors could be *Architectural component behavior* which describe the behaviour or technicalities of components, regardless of whether they are small or large, within the architecture at hand [6]. These decisions factors are used to all sentences such that these annotations can be used for further quantitative analysis.

Issue	**Example**
HDFS-2802	*"Low resource overhead at the NN to maintain snapshots"*
HADOOP-3750	*"Actually, the dependency should just be: hdfs depends on core, mapred depends on core"*

   The sentence above gives a clear indication of the decision factor involved. The first sentence would be labeled as a *Quality Attribute Requirement* and the second sentence would be labeled as the *Architectural Design Configuration* concept.

Issue	**Example**
HDFS-3750	*"I think this patch can go in and doesn't need to wait for HADOOP-4631. Checking this in early would restrict users to introduce any cyclic dependency in code."*

   Not every comment or sentence provided useful information. As shown in the examples above, some sentences are too technical or have nothing to do with the decision factors described in the coding book by Soliman et al.

3. **Quantitative Analysis of the annotations**
After the annotation process qualitative analysis is performed to answer our research questions. **Research Question 1** and **2** will be answered by annotating all issues as described above and filter out any issues that do not contain valuable content. Issues are considered to not be useful when they contain either no long discussions or contain discussions on a technical level that is too low to be of interest. An example of this would be extensive lists of test coverage or results.

   Research Question 3 is answered by analysing the number of times a certain concept is used in combination with another concept. The significance of the co-occurrence is calculated using $\tilde{\chi}^2$.

   Research Question 4 will be answered by revisiting all annotations and determining the form for the accompanying AK-concept for every single one of the annotations in the set. These forms are found in the coding book and are used to determine the distribution of variance. The last research question, **Research Question 5** is answered following the next steps:

   (a) Selected the most common AK-concepts based on set of concepts found while answering step 1.

   (b) In order to determine the taxonomy or most frequently used words for each concept there are a few important concepts to consider. The first step is to remove English stop words as these should not count towards the total. Using the NLTK library for Python this can be achieved without any issue [9]. NLTK is widely known as a tool that is useful when working with language data. It provides multiple functionalities which can be used to process text to be used for further analysis. One of the features of NLTK that is important to be considered during the study is Lemmatization. Lemmatization is the process of converting every word into the unique form of the word. For example, the words *am*, *are* or *is* all converge into the verb *be*. When determining the taxonomy of a piece of text, Lemmatization should be used such that all text is properly classified.

5 Results

5.1 Distribution

During the annotation process over 100 issues and attachments were analyzed, resulting in over 3500 annotations which serve as the data set for this research. All annotations are categorized according to whether they belong to the description of the issue, the comment section or are found within an attachment.

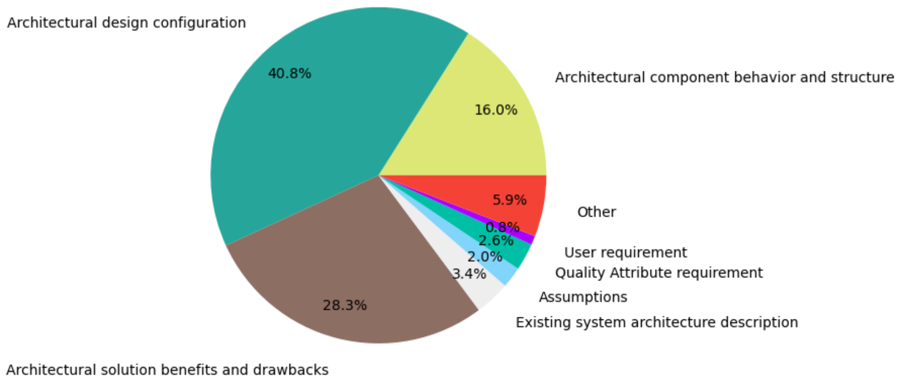

Figure 5.1: *Distribution of different concepts in the issue description.*

As shown in Figure 5.1 the greater part of the description contains ($> 85\%$) either Architectural Component Behaviour and Structure, Architectural Design Configuration or Architectural Solutions benefits and drawbacks. A reason for this could be that the description of an issue is often used to propose a solution for a certain problem or requested feature. This suggestion is supported by the fact that Run-time Quality Issues ($\approx 3\%$) and the Existing System Architecture Description ($\approx 6\%$) are also present relatively often.

When comparing the distribution of the comment section to the description there are no significant changes noticeable other than that the concepts Run-time Quality Issues and Existing System Architecture Description occur less frequently.

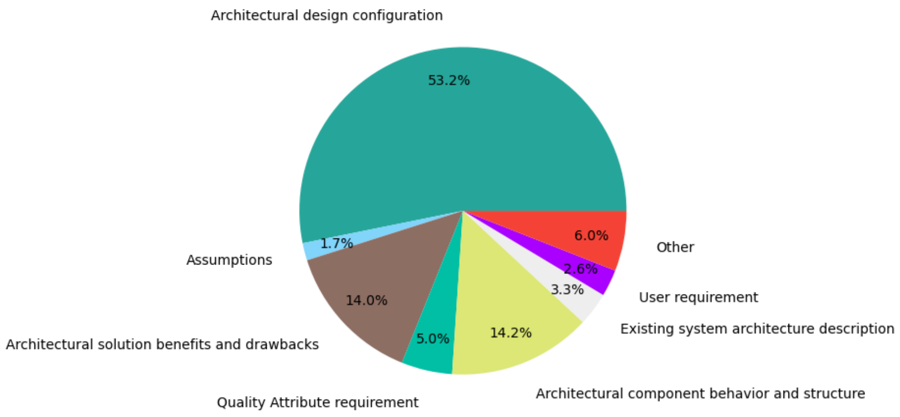

Figure 5.2: *Distribution of different concepts in the attachments.*

Although there are no major differences between the issue description and the comment section of the issues there is a noticeable difference present when looking at the attachments. Where the Architectural

Design Configuration made up less than half of the annotations ($\approx 40\%$) in the description and comments, almost 60% of the annotations within the attachments focuses on the design configuration. Together with Architectural Component Behaviour (12.5%) and solution benefits and drawbacks (17.5%) there are three concepts that make up most of the concepts that are discussed. This suggests that the attachments are mostly used to substantiate the design decisions that are discussed in the comment section.

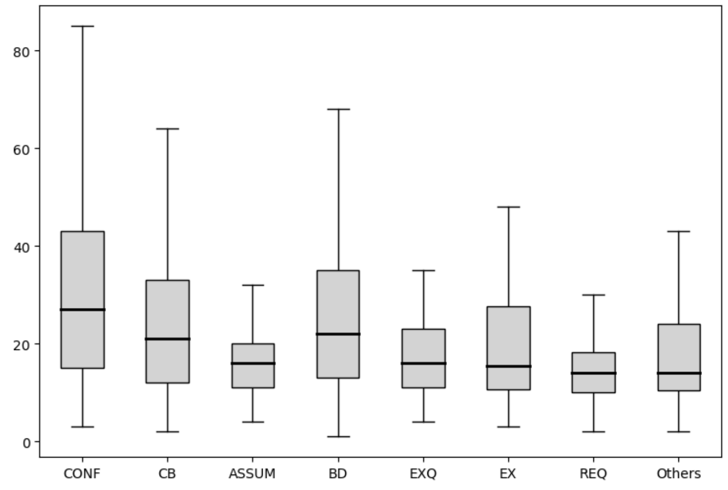

Figure 5.3: *Distribution of the lengths per annotation-size for the most common AK concepts.*

Looking at the annotation size in Figure 5.3 it becomes clear that the issues that are the most common concept among the annotations are also written about in a longer form. This shows that discussions about the architecture, the concepts behind it and it's benefits and drawbacks are of greater importance then other AK concepts.

5.2 Variants

Each AK Concept can be described in different forms. These various forms make it easier to recognize a certain concept. For the four most common AK concepts (>90%) the textual variants are reviewed in greater detail. In the results below the description, comment section and attachments are grouped before calculating the variance of a certain concept. There is only a small percentage difference ($< 5\%$) on each variant when looking at the different sections which is therefore not of interest.

5.2.1 Architectural Component Behaviour and Structure

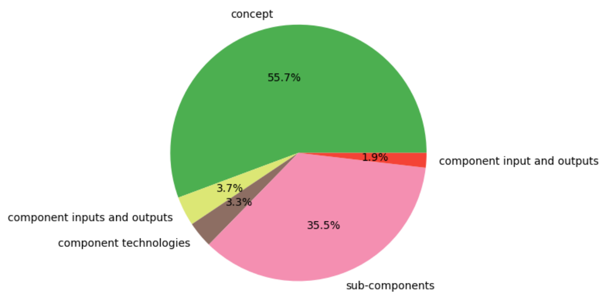

Figure 5.4: *Textual variants of the Architectural component behaviour and structure concept.*

Figure 5.3 shows the distribution of the various forms the *CB* has been identified as. The concept appears to be most often described by describing the high-level concept ($\approx 55\%$) behind the proposed solution such as the behaviour or approach to solve a problem [6].

5.2.2 Architectural Design Configuration

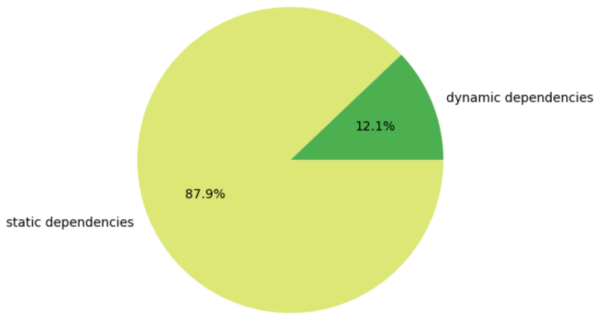

Figure 5.5: *Textual variants of the Architectural Design Configuration concept.*

Figure 5.4 shows the distribution of the various forms the *CONF* concept concept has been identified as. The concept appears to be most often described by describing static dependencies, or, in other words, what components are involved in creating the proposed solution [6]. Dynamic dependencies are described less often indicating that the exact order in which components interact are not that important when discussing the initial solution.

Issue	**Example**
Hadoop-2184	*"Is it possible to have a situation that a RPC socket is shared by two identities?"*
Hadoop-2184	*"Maybe we should just add a Ticket field to Invocation. Then the Client can pass the proxy's ticket with the call, as an invisible parameter. "*

The Design configuration describes either static or dynamic dependencies. However, it also happens that questions are asked regarding the design configuration or that suggestions are being made as can be seen in the table above. Questions or suggestions do not naturally fit in the category of static or dynamic dependencies as the intention behind it is quite different since they contain an uncertainty.

5.2.3 Solution Benefits and Drawbacks

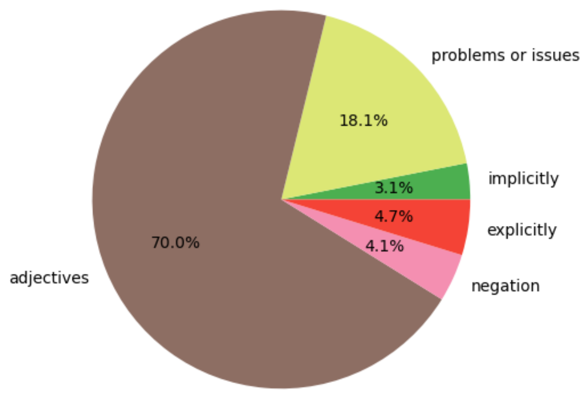

Figure 5.6: *Textual variants of the Architectural Solution Benefits and Drawbacks concept.*

Figure 5.5 shows the distribution of the various forms the *Architectural Solution Benefits and Drawbacks* concept has been identified as. Benefits and drawbacks are in most cases (≈70%) described using adjectives such as "good" or "bad" [6]. Problems or issues are also (≈ 20%) used regularly to describe why a proposed solution is not optimal or can be improved.

5.2.4 Existing System Architecture

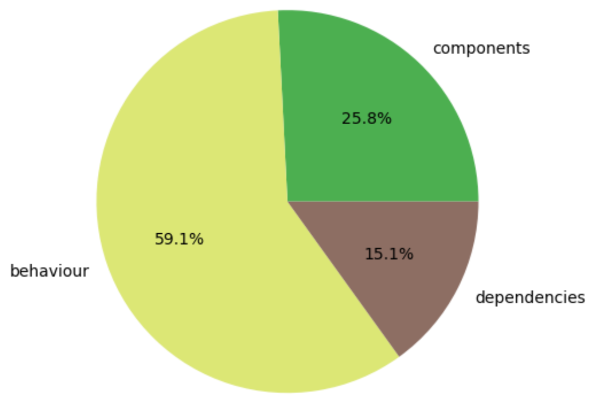

Figure 5.7: *Textual variants of the Existing System Architecture concept.*

Figure 5.6 shows the distribution of the various forms the *Existing System Architecture* concept has been identified as. In a large part of the annotations (≈ 60%) we found that when the existing architecture was discussed, the behaviour of the system was more important than the actual implementation on a lower level. A reason for this could be that the current behaviour often needs to be improved or build upon in which case the behaviour receives the most attention.

A differentiation can be made between describing the existing architecture and components but also the benefits and drawbacks of the current implementation. Benefits and drawbacks of the current system are often an important factor when making decisions regarding new changes or features and it can be argued that this should be another textual variance of the existing system architecture.

Issue	**Example**
Hadoop-4952	*"Today a site.xml is derived from a single Hadoop cluster. This does not make sense for multiple Hadoop clusters which may have different defaults."*
Hadoop-7240 Attachment	*"HDFS does the following very well: Scaling (storage, IO, clients Horizontal scaling – IO + PBs), Fast IO – scans and writes,Number of concurrent clients 60K to 100K++ ..."*

5.2.5 Assumptions

Often times within a discussion the preference for a certain solution is expressed. However, these preferences are expressed without any supportive arguments meaning that it is simply a form of personal preference. The preference would fit in the "Assumptions" concept but is not described explicitly nor using uncertainty-related terms. A new textual variance to be added to *Assumption* could be called 'Implicit' or 'Based on preference'.

Issue	**Example**
Hadoop-19285	*"My preference is this feature, like all scan features, should be outside the NN."*

5.3 Co-occurrence

Figures 5.8 and 5.9 show the co-occurrence between different AK concepts indicating the relation between the concepts. The thickness of the lines connecting the different concepts shows how often the different concepts are present together within the specified parts of the issues. This size of each of the nodes indicates how often the same concept is present multiple times next to each other.

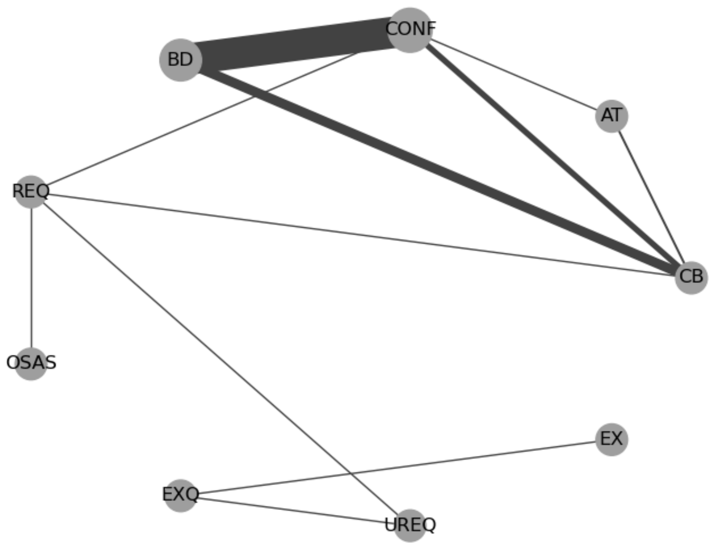

Figure 5.8: *Co-occurrence between concepts in the description and comments.*

The description shows a clear presence of relations between ABD, CONF and CB. The design configuration also co-occurs with a lot of other concepts.

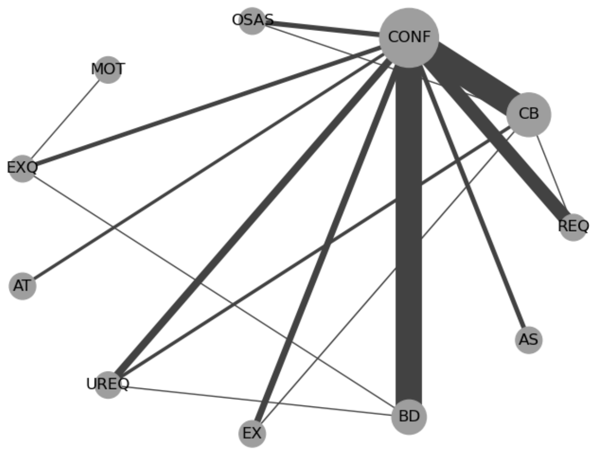

Figure 5.9: *Co-occurrence between concepts in the attachments.*

Comparing the co-occurrence within the description and comment section to that of the attachments there are similarities as well as differences. The biggest similarity is the strong presence of co-occurrence between the concepts BD, CONF and CB, meaning these concepts are key factors regardless of their position. The biggest difference is the lack of co-occurrence between concepts in the description and comments sections where they are present in the attachments.

5.4 Taxonomy

Based on the most commonly used AK concepts found in section *5.1* we will take a closer look at the taxonomy of the ten most common words used in the issues for each separate concept.

Rank	Word	Occ.
1	file	94
2	block	62
3	data	61
4	need	61
5	use	60
6	key	58
7	one	55
8	node	54
9	method	54
10	also	53

Rank	Word	Occ.
1	block	79
2	container	52
3	namenode	50
4	directory	49
5	state	47
6	storage	46
7	token	46
8	data	37
9	file	34
10	current	32

Figure 5.10: *Taxonomy of the Architectural Component Behaviour and Structure concept in the description and comments (l) and the attachments (r).*

When looking at the most common words for the Component Behaviour concept it is interesting to see the difference between the description and the attachments. Were the most common words in the comment section are not very specific a distinction can be made when looking at the words from attachments. Words such as *block*, *directory* or *container* indicate the architectural components which are involved in the issues and are used rather often.

Rank	Word	Occ.
1	block	312
2	file	231
3	would	219
4	client	214
5	user	205
6	data	199
7	need	194
8	use	159
9	nn	156
10	also	148

Rank	Word	Occ.
1	data	328
2	block	324
3	storage	216
4	hdfs	212
5	file	207
6	client	198
7	namenode	183
8	user	173
9	key	160
10	node	160

Figure 5.11: *Taxonomy of the Architectural Design Configuration concept in the description and comments (l) and the attachments (r).*

When comparing the taxonomy of issues for CONF and CB there are not a lot of differences between the most common words in the description and comments, nor in the attachments. This makes sense considering both concepts are related to describing the proposed solution to architectural changes.

Rank	Word	Occ.
1	would	150
2	block	115
3	also	111
4	data	109
5	file	107
6	client	94
7	hdfs	94
8	like	93
9	since	93
10	could	90

Rank	Word	Occ.
1	hdfs	38
2	need	34
3	block	33
4	client	32
5	data	28
6	file	28
7	storage	28
8	token	26
9	the	26
10	namenode	24

Figure 5.12: *Taxonomy of the Architectural Solution Benefits and Drawbacks concept in the description and comments (l) and the attachments (r).*

The Solution Benefits and drawbacks do show a difference however, when being compared to the taxonomy of either the CONF or CB concept. The description and comments contain words such as *would*, *also*, *since* or *could* which are commonly used to describe benefits or drawbacks. The attachments show no major differences compared to the other AK concepts, probably because they are often used to help as a supportive piece for the rest of the discussion on the issue itself.

6 Discussion

Multiple issues contained comment sections that focused on testing or were simply too low-level. In multiple issues, the implementation details of a certain method are discussed. Although these implementations are supported by design decisions somewhere earlier in the development life cycle, the actual implementation details are not as relevant for this study as they do not provide insights in the architectural design decisions made. Further research could be done in order to automatically rule out the issues that do not contain relevant information.

6.1 Research Question 1:

Issues contain a lot of information on architectural design decisions made during the development process. Although there are more than 10 different AK concepts that can be distinguished when looking at the issues, only a small subset of those is used regularly as can be seen while looking at figure 5.1. CB, CONF and BD combined form over three quarters of the annotations found. This indicates that the architectural design and structure of the system and its components are oftentimes of greater importance and that advantages and disadvantages are used to discuss these architectural designs.

One of the risks in determining that the issues are following the above mentioned approach by default is that only two different projects were analyzed during this study. Although both projects were completely separate part of the system, often times with different people working on them, they still belonged to Apache Hadoop. This could mean that when looking at completely different projects, the main focus throughout the development process could be centered around other AK concepts. Due to this uncertainty, further research could look into this in order to get a better overall understanding of the distribution of AK concepts.

6.2 Research Question 2:

Looking at the results presented in section 5.1 it becomes clear that a differentiation can be made between the issue itself, consisting of the description and the comment section, and the attachments. The attachments are more centered around the technicalities and structure of the architecture rather than supporting the decisions that lead up to them.

The results show that the description and comment section contain more information about the potential benefits and drawbacks than the attachments. The attachments contain more information about the architecture and components involved as well as statements regarding the Quality Attributes. This knowledge can be used during further research depending on the focus of the study. If recovery of the architecture is more important than the reasoning behind it, the attachments might provide better value.

6.3 Research Question 3:

The co-occurrence found during this research shows that there is a lot of information that can be found within the issue. Concepts such as Architectural Design Configuration, Benefits and Drawbacks and Component Behaviour often co-occur which can be useful when searching for AK in new issues.

The attachments show a stronger co-occurrence between multiple different

AK concepts. However, since there were less attachments than comments and descriptions and they were shorter in general as well, less annotations could be made in the attachments. This could mean the occurrence is a bit skewed.

Further research could focus solely on attachments in order to gain more insights in the different AK concepts present and the relations between them.

6.4 Research Question 4:

While analyzing the textual variants it became apparent that the different forms of the AK concepts were not always sufficient. During the classification process multiple new forms arose which can be seen as an extension of the existing coding book. The existing system architecture concept currently does not describe a form to indicate the benefits and drawbacks of the existing system are mentioned, this variant should be added to the coding book.

6.5 Research Question 5:

The taxonomy found in the issues provides an insight in the words commonly found for each concept. The results show the attachments provide less value when focusing on the taxonomy. In general, the attachments play a more supportive role for the issue itself. This means the taxonomy does not differ a lot when comparing different AK concepts. On the other hand, the comment and description provide different taxonomies depending on the AK concept.

To improve the usability of the taxonomy, one could utilize Machine Learning algorithms to train models on recognizing the AK concept based on the taxonomy of the sentences or issues. As an example, the Solution Benefits and Drawbacks often use words as *also* or *since* that are used to support a certain statement that explains the advantage or disadvantage of a proposed solution. Knowing when these words are used and how they relate to specific concepts could be determined by further research.

7 Threats to validity

As only two different sub projects were used during this research study the results might not represent all real world projects. Even within the projects considered in this study there were differences in terms of writing and communication style. In order to get a more representative view of the presence of AK concepts within issue tracking systems the research should be expanded to include projects of different development areas and projects.

Similar research done on the same subject is still quite different from our study. Other studies make use of different models, tools or AK concepts and therefore our results are difficult to verify. In extension to this, since the annotations were largely created by one person only the results could be biased. This could be improved upon by having other researchers validate the annotations.

8 Conclusions & Future work

The goal of this study is to capture present architectural knowledge present in issue tracking systems. Within the set of issues we were able to determine the distribution of the different AK concepts present. The co-occurrence between AK concepts and the taxonomy of every concept, gives an insight in how the sentences are related and to the AK concept they belong to.

Using the knowledge that has been gathered, further research can be done such that AK concepts can be identified and reused at a larger scale. One of the areas that could be researched further is the field of taxonomy. If more data is collected and a structure behind the sentences can be derived based on the different AK concepts this could pave the way for automatic capturing of these concepts.

As an extension of this study, further research could involve trying to capture the concepts that support another concept, somewhat similar to the co-occurrence but focusing on the explicit order of the concepts used. For example, it could be the case that most architectural design decisions are being decided upon after solutions and benefits are discussed. This would involve statistical analysis as well as additional labeling done by the researcher.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (SEI Series in Software Engineering).* Addison-Wesley Professional, 4 ed., 2021.

[2] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," *Quality of Software Architectures*, pp. 43–58, 2006.

[3] A. Jansen, J. Bosch, and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *Journal of Systems and Software*, vol. 81, no. 4, pp. 536–557, 2008.

[4] A. Shahbazian, Y. Kyu Lee, D. Le, Y. Brun, and N. Medvidovic, "Recovering Architectural Design Decisions," *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018.

[5] M. Shahin, P. Liang, and Z. Li, "Recovering software architectural knowledge from documentation using conceptual model," *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2013, 06 2013.

[6] M. Soliman, M. Galster, and P. Avgeriou, "An exploratory study on architectural knowledge in issue tracking systems," 2021.

[7] A. Tang and H. van Vliet, "Software Architecture Design Reasoning," *Software Architecture Knowledge Management*, pp. 155–174, 2009.

[8] "Apache Hadoop."

[9] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python.* Culemborg, Netherlands: Van Duuren Media, 2009.

A Appendix

The following section contains an overview of the taxonomy for multiple AK concepts discussed in this paper.

A.0.1 Taxonomy Architectural Design Configuration (l) and taxonomy Solution Benefits and Drawbacks (r)

Rank	Word	Occurrences
1	block	312
2	file	231
3	would	219
4	client	214
5	user	205
6	data	199
7	need	194
8	use	159
9	nn	156
10	also	148
11	hdfs	143
12	storage	141
13	one	135
14	new	129
15	could	123
16	files	119
17	rpc	110
18	node	109
19	case	107
20	namenode	102
21	think	101
22	make	100
23	like	100
24	read	97
25	dn	90
26	datanode	89
27	directory	88
28	call	84
29	method	82
30	interface	82
31	implementation	82
32	may	82
33	want	82
34	snapshot	82
35	get	81
36	key	81
37	state	81
38	sps	81
39	since	80
40	task	79
41	using	79
42	class	79
43	cluster	79
44	job	78
45	change	77
46	system	77
47	support	76
48	code	74
49	way	74
50	operation	74

Rank	Word	Occurrences
1	would	150
2	block	115
3	also	111
4	data	109
5	file	107
6	client	94
7	hdfs	94
8	like	93
9	since	93
10	could	90
11	need	90
12	user	87
13	think	86
14	use	81
15	one	77
16	case	75
17	make	71
18	code	69
19	may	66
20	new	64
21	node	61
22	way	61
23	work	60
24	read	57
25	approach	56
26	get	54
27	snapshot	54
28	nn	54
29	change	53
30	disk	51
31	storage	51
32	hadoop	50
33	might	50
34	files	49
35	much	49
36	time	48
37	key	48
38	using	47
39	problem	46
40	number	45
41	cluster	44
42	system	44
43	want	43
44	support	43
45	1	42
46	namenode	42
47	class	42
48	issue	41
49	layer	41
50	add	41

A.0.2 Taxonomy Architectural Component Behaviour (l) and taxonomy Existing System Architecture Description (r)

Rank	Word	Occurrences
1	file	94
2	block	62
3	data	61
4	need	61
5	use	60
6	key	58
7	one	55
8	node	54
9	method	54
10	also	53
11	would	52
12	files	50
13	path	50
14	archive	49
15	rack	49
16	name	48
17	new	47
18	int	46
19	value	46
20	job	45
21	public	45
22	user	44
23	cache	43
24	directory	43
25	like	43
26	could	41
27	class	40
28	interface	35
29	hdfs	34
30	map	34
31	list	32
32	client	32
33	support	32
34	get	31
35	think	31
36	change	29
37	partition	29
38	storage	29
39	make	28
40	code	28
41	string	28
42	bufer	28
43	case	27
44	output	27
45	default	26
46	byte	26
47	using	25
48	spill	25
49	object	25
50	nn	25

Rank	Word	Occurrences
1	currently	25
2	hdfs	24
3	archive	20
4	data	19
5	dfs	17
6	block	17
7	file	16
8	job	15
9	current	15
10	hadoop	12
11	namenode	12
12	client	12
13	user	11
14	cache	11
15	access	10
16	task	9
17	code	9
18	use	9
19	read	9
20	layer	9
21	implementation	9
22	two	8
23	localized	8
24	path	8
25	get	8
26	also	8
27	run	8
28	already	8
29	rpc	8
30	method	8
31	storage	8
32	directory	7
33	mapred	7
34	would	7
35	running	7
36	using	7
37	local	7
38	support	7
39	node	7
40	interface	7
41	system	7
42	datanode	7
43	policy	7
44	mechanism	6
45	jar	6
46	cluster	6
47	set	6
48	list	6
49	name	6
50	information	6

A.0.3 Taxonomy Quality Attribute Requirements (l) and taxonomy User Requirements (r)

Rank	Word	Occurrences
1	performance	14
2	block	10
3	read	9
4	client	7
5	compatibility	7
6	storage	7
7	like	7
8	need	6
9	make	6
10	take	6
11	would	6
12	one	6
13	time	5
14	rpc	5
15	multiple	5
16	data	4
17	user	4
18	number	4
19	interface	4
20	server	4
21	jira	4
22	using	4
23	improvement	4
24	hdfs	4
25	design	4
26	snapshot	4
27	deployment	4
28	caching	4
29	let	3
30	cache	3
31	support	3
32	reduce	3
33	operation	3
34	purpose	3
35	replica	3
36	improve	3
37	availability	3
38	write	3
39	latency	3
40	2	3
41	backwards	3
42	serialization	3
43	files	3
44	must	3
45	consider	3
46	requirement	3
47	since	3
48	higher	3
49	throughput	3
50	layer	3

Rank	Word	Occurrences
1	user	10
2	files	7
3	fle	7
4	policy	7
5	use	6
6	storage	6
7	using	5
8	operation	5
9	one	5
10	admin	5
11	key	4
12	join	4
13	want	4
14	system	4
15	feature	4
16	requirement	4
17	directory	4
18	hdfs	4
19	job	3
20	code	3
21	record	3
22	i/o	3
23	application	3
24	like	3
25	important	3
26	mapping	3
27	hadoop	3
28	see	3
29	drive	3
30	node	3
31	archive	2
32	program	2
33	avoid	2
34	3	2
35	case	2
36	many	2
37	library	2
38	implement	2
39	people	2
40	support	2
41	ability	2
42	sort	2
43	thing	2
44	would	2
45	admins	2
46	–	2
47	may	2
48	comparator	2
49	class	2
50	property	2