Exploring the Rationale of Design Decisions in Open-Source Software Mailing Lists and their Relationship to Architectural Issues

MSc. CS Research Internship (2021–2022) – W. Meijer (s4509412)
Supervisors: dr. M. Soliman and prof. dr. ir. P. Avgeriou
University of Groningen – August 30, 2022

***Abstract*—In open-source software, architectural knowledge (AK) is incoherently spread across various platforms such as issue tracking systems, source code and mailing lists. Previous research has explored AK concepts discussed in issue tracking systems and blogs. However, no similar exploration has been performed on mailing lists yet. Similarly, the relationship between issue tracking systems and mailing lists is not yet done. This study attempts to bridge this gap by exploring what decision rationale types are used in mailing lists and by identifying how architectural issues are used in mailing lists. To identify rationale types and architectural email-issue relationships, thematic analysis is used on emails sampled from six Apache projects: Cassandra, Tajo, and the four sub-projects of Hadoop: Hadoop-common, HDFS, MapReduce, and Yarn. To determine what decision rationale is used in mailing lists and what its relationship is with decision types, an analysis is performed on 156 architectural emails, identifying nine types of decision rationale, six relationships between rationale types and three relationships between decision types and rationale types. To identify architectural email-issue relationships an analysis is performed on 482 architectural emails, identifying three relationship superclasses, six relationship subclasses and 14 relationships between relationship types. The findings of this work create a better understanding of how mailing lists complement other sources of AK. This allows practitioners to better navigate the vast amount of AK spread across these sources and opens up paths for researchers regarding automated AK identification and synthesis.**

***Keywords*—Architectural Knowledge, Decision Rationale, Open-Source Software, Mailing Lists, Issue Tracking Systems, Deductive Thematic Analysis, Inductive Thematic Analysis.**

I. Introduction

Architectural knowledge (AK) is defined as the different structures and components used in the concrete architecture of software, combined with the decisions and rationale used to acquire these structures [1], which is commonly manifested in the early design decisions of a development process [2]. AK can therefore be considered as the cornerstone of any software system, as this contains the most important information of any given architecture. However, due to a lack of resources, a lack of urge or simple oversight, a lot of architecturally relevant information goes undocumented for which it remains tacit in the heads of software architects [3]. This is an especially important problem in open-source systems as participants of these systems are very commonly distributed across geographical and temporal zones for which direct communication is often very difficult. Even though documentation is very important, Ding *et al.* [4] found that only 5% of all open-source systems have some form of architecture documentation – in other words, the vast majority of open-source systems do not.

In these systems, various sources of information exist, such as issue tracking systems, mailing lists, or source code. Potentially, these different sources can be used to retrieve valuable pieces of AK that could be used during software evolution or when creating completely new systems. Developers attempt to use these sources, combined with forums and general-purpose search engines, to retrieve AK, however, they find it difficult to locate AK like decisions and quality concerns [5]. Therefore, finding better alternatives that can be used to answer architectural issues remains an important topic of research.

The work performed by Mannan *et al.* [6] showed that 89% of all decisions made in open-source software were made on their respective mailing list. Therefore, mailing lists are a potential gold mine containing insights into the decision-making process of open-source software. Whereas other sources of information have been explored to some extent already [7], [8], mailing lists remain relatively unexplored in literature. The work presented here is an exploratory study, attempting to gain deeper insights into the AK shared in mailing lists of open-source systems. This work extends the work of Lalis [9], who identified decision types [1] in mailing lists, by identifying what decision rationale types are used to make architectural decisions. Additionally, the work performed by Soliman *et al.* [7] and Faroghi [10] will be extended by identifying how architectural issues are used in architectural emails. Therefore, the goal of this research is defined as follows:

The goal of this work is to identify what rationale types are used for architectural decisions in mailing lists of open-source software and what the relationship is between architectural emails and architectural issues in open-source software.

The rest of this document is organised as follows. First, the necessary theoretical background and related work are discussed in section II. Section III introduces the posed research questions, the data that is analysed, and the used research methodology. The results of this study are presented in section IV, followed by a discussion of these accompanied by their implications on practitioners and researchers in section V. Section VI introduces a number of threats to the validity of this study as well as how these were mitigated. Finally, section VII describes the conclusions that can be drawn from this work.

II. Theoretical Background

The field of AK management has existed for over a decade already, for which several related studies have been performed already. Section II-A sheds light on the fundamental theory of AK, emphasising architectural decisions and decision rationale. This is followed by section II-B, describing how mailing lists can be used as a data source. Finally, section II-C introduces several related studies on AK classification and issue relationships, elaborating how this work complements those studies.

A. Architectural Knowledge

Software architecture is commonly considered to be the sum of the architecture itself (i.e. the product) and the architectural

decisions that lead to the creation of that architecture (i.e. the process of generating the product) [1]. The architecture alone cannot be considered sufficient information as the reason for a solution could explain whether change is allowed or if that will introduce system regressions. Much of the rationale behind the architecture is tacit and evaporates over time, introducing the risk of architectural decay [11]. Because of this, architectural decision-making has grown as a separate field of research and various models have been created to capture the different components of such a decision. The works of Zimmermann *et al.* [12], [13] describe architectural decisions in three distinct components: an issue, solution alternatives, and an outcome. Here, the issue describes a problem within the system, solution alternatives provide a means of solving that issue, and outcomes describe what the chosen solution is combined with the rationale for that decision. Although this model captures core components of architectural decision-making, reality shows that architectural decisions are commonly not documented in such a systematic manner [4].

Decisions are made on different abstraction levels [14], use different levels of cognitive effort [15], and address different types of problems [1]. The work of Ven *et al.* [14] describes that high-level decisions commonly address general issues such as the general architecture of the system (e.g. service-oriented architectures) and low-level decisions are generally so specific that they have no significant impact on the system as a whole. They argue that medium-level decisions have the highest influence as they generally discuss components, frameworks, or solution patterns. Decisions can be made naturalistic and rationalistic [15], describing the level of cognitive effort put into making decisions. Whereas rational decision-making attempts to explore all solution factors and commonly multiple alternatives, naturalistic decision-making commonly relies on previous experience, making it vulnerable to biases and fallacies.

The work of Kruchten *et al.* [1] provides insights into different types of decisions. In their work, four fundamental decision categories are described: *Existence, Non-Existence (Bans), Property*, and *Executive Decisions*. *Existence decisions* address the components of a solution and can be subdivided into *Structural* and *Behavioural* decisions, respectively describing the structure of the architecture and the behaviour between components. *Non-existence decisions* are the inverse of this class, describing the structure and behaviour the system will not have. *Property Decisions* indicate what properties the system fulfils such as quality requirements or quality attribute prioritisation. Finally, *Executive Decisions* describe the development process and are subdivided into *Process, Technology*, and *Tool Decisions*. Respectively, these address the development process (e.g. agile development), what technologies are used (e.g. Java libraries), or what development tools are used (e.g. an IDE). The work of Soliman *et al.* [8], [16] expands on the definition of decisions, introducing *Decision Recommendations*. Rather than being a conclusive decision, decision recommendations merely suggest a preference for a solution alternative. In environments with a more democratic structure (like projects analysed in this study), the various recommendations can eventually lead to a binding decision. Although decision recommendations are not included in the works of Kruchten *et al.* [1] and Zimmermann *et al.* [12], [13], it is a natural extension of these works as a decision recommendation is simply a non-binding variant of a decision.

Decision rationale, the reason why a decision is made, has been explored in the literature as well. Tang *et al.* [17] present a means to systematically address design reasoning by describing why someone should be concerned about this and how they can do this. Additionally, they identify various types of rationale that are relevant to this study: *Constraints, Benefits, Trade-offs, Risks*, and *Assumptions*.

Within software architecture, *Constraints* are defined as a limiting factor that specifies what conditions a new solution must adhere to for it to be considered a viable option [17]–[20]. Constraints can arise from requirements or previously made decisions. Constraints can be both soft and hard in nature, meaning that although they should be fulfilled, it is not always a necessity that it is. An example of this is a user requirement versus the compatibility constraints imposed by an adopted technology – one could choose to ignore the former whereas the latter must be respected.

Benefits describe the strengths of a solution alternative and commonly go hand-in-hand with *Drawbacks* because the benefit of one solution is commonly the drawback of another [7], [16], [21]. *Trade-offs* describe how solutions balance each other out in terms of their qualities. Commonly, trade-offs are weighted according to their priority within the system. An example of this is implementing an encryption algorithm, which improves security but reduces performance because security has a higher priority than performance. Trade-offs can be made on both a high and a low level. High-level trade-offs indicate the general prioritisation of system qualities, while low-level trade-offs describe the benefits and drawbacks of a concrete solution. Consequently, high-level trade-offs do not always directly imply a concrete benefit or drawback.

The work of Yang *et al.* [22] expands on *Assumptions* as rationale, defining it as *"AK taken for granted, or accepted as true without evidence"*. Various literature has introduced definitions for *Risks* [2], [17], [23]–[25]. Based on these definitions, this work defines *Risks* as follows: *"Risks are potential undesirable consequences on the software architecture of a system in light of stated quality attribute requirements. Risks have characteristics such as impact, probability, mitigation time frame, coupling, and uncertainty"*. A commonly used counterpart of *Risks* is *Non-Risks*, which are *Risks* that are deemed safe after observation. *Assumptions* and *Risks* share the component of uncertainty, making them similar classes. However, the subtle difference between these is that it is unclear whether *Assumptions* are true, whereas *Risks* are true but simply not always active.

Although the list of Tang *et al.* [17] is comprehensive, various other types of rationale, relevant to this study, have been defined in the literature. The work of Soliman *et al.* [8], [21] introduces *Solution Evaluation*, which describes how solution alternatives are or should be evaluated (e.g. a time performance benchmark). In their ontology Soliman *et al.* [21] also introduce *Solution Comparison*, describing a comparison between two architectural solutions based on their architecturally relevant capabilities. Finally, Soliman *et al.* [16] expand on *rationale* concepts again, introducing *Decision Rules*. Such rules take the form of *"if A then B"* where A is a condition, and B is a consequence. These rules impose specific requirements on a solution alternative, specifying what a solution should adhere to.

In some cases, the fact that a given problem is tackled is deemed as sufficient rationale to make changes to the system. This is particularly the case when *Quality Issues* are addressed. *Quality Issues* are a special type of issue that address a problem related to a quality attribute (e.g. performance). Various *Quality Attributes* have been defined in the literature (e.g. ISO/IEC-25010 [26]), however, any system can define quality attributes at its discretion [2], for which there exists no exhaustive list of them. Because *Quality Issues* describe a problem within the system, their mere existence can be used to apply changes to the system (as a system benefits from having fewer problems).

Currently, no exhaustive and widely-accepted ontology for rationale types exists yet and not all of the established rationale types are relevant to this study, the list of rationale types that is used in this study is composed of various pieces of literature [2], [7], [8], [16]–[25].

B. Mailing Lists and Issue Tracking Systems

Open-source software has many potential sources of AK. Together with issue tracking systems and blogs, mailing lists are one of these sources. In open-source systems, mailing lists are used to share various types of information related to the system, such as release announcements, new solution proposals, or interesting events. Because of this, mailing lists contain many different emails, discussing many different topics, wildly varying in architectural relevance. Emails consist of some components: a subject (the title), a body (the actual message), the sender/receiver, and the date on which it was sent. Information shared inside emails is generally not structured according to any given format, it is not clear what type of information is shared in them.

Mailing lists can be observed in two ways: a set of individual emails, and a set of email threads. Mailing threads differ from individual emails as it considers an email and all of the replies to it to be one entity. This is somewhat analogous to an entire conversation (a mailing thread) versus observing individual remarks (an individual email). It is intuitive to say that emails contained in the same mailing thread are more likely to discuss the same topic than two randomly picked emails, for which it is much easier to understand the topics that are discussed.

Within the Apache community, mailing lists are most commonly used to ask questions, initiate discussions, provide updates, cast votes, or make higher-level decisions. Conclusions reached in these conversations commonly continue in issues stored in issue tracking systems (e.g. Jira or Github), where lower-level details are discussed. Concrete solutions discussed here are then implemented, after which they are evaluated through pull requests. One of the benefits of using mailing lists is that, compared to issues on issue tracking systems, many more people receive these emails which is why (generally) more important topics are discussed in mailing lists.

Issue tracking systems are a popular tool to track a software's health, by tracking all of its *issues*. An issue can describe tasks, new features, bugs, or user stories, generally described using natural language. Similar to emails, issues are outfitted with a title, a body, and a creation date. However, differently compared to emails, issues identify a reporter, an assignee (experts related to this issue), as well as tags (e.g. *"bug"* or *"improvement"*), their status (e.g. *"solved"*), and an ID. This ID is commonly used in other communication platforms to establish a link with the respective issue. An example of this is *"I'd like to talk about CASSANDRA-10993"*, where the Issue ID is referenced in an email on the *Cassandra* mailing list.

Throughout this work, the notion of *Architectural Emails* is used various times. *Architectural Emails* are emails in which AK [1] is shared. Although this is a rather straightforward definition, identifying whether an email is architectural is difficult. To simplify this, the definition of *Architectural Email* used in this work is *"Emails that shed light on architectural components or architectural decisions (such as used technologies or structural changes) [1], giving concrete insights into decision components (such as the problem description or solution rationale) [12], [13], including recommendations for design decisions [16], are considered to contain architectural significant information"*.

C. Related Work

Multiple studies related to this work have been performed in the past. This section introduces some studies related to manual and automated AK classification, issue relationships, and how this work diversifies itself from these studies.

1) AK Classification:

The work of Soliman *et al.* [7] performed an exploratory study on architectural decisions made in issue tracking systems. Through deductive thematic analysis, using AK concepts introduced in previous literature, they were able to identify three types of concepts: *decision factors, architectural solutions, and decision rationale*. Their results showed that architectural issues most commonly contain solution descriptions with corresponding decision rationale, rather than decision factors. In turn, solution descriptions most commonly address component behaviour and component configuration, and the rationale most commonly described benefits and drawbacks and assumptions. Furthermore, they explored the co-occurrence of AK concepts in architectural issues, identifying various relationships between them. Another work of Soliman *et al.* [8] explores technology decisions in StackOverflow. They find that AK in developer communities consists of two dimensions: the *purpose* and *solution* dimensions. Here, the purpose describes the functionality of the discussed technologies, attempting to determine whether said technology matches the required functionalities. In turn, the solution type emphasises the feature set of the different technologies, the purpose of the technology, or how the discussed system's component configuration can account for a specific technology.

The work performed by Xiong *et al.* [27] explores the types of assumptions made inside the Hibernate mailing list. They classified these assumptions into four different assumption types: *Requirement Assumption, Design Assumption, Construction Assumption*, and *Testing Assumption*, which are subclassified into ten different subclasses. Their results show that over 80% of the made assumptions are design and requirement assumptions. Finally, the work performed by Li *et al.* [28] explores decision-making in mailing lists, identifying the decision type and decision rationale. They found that the decisions made in mailing lists are of five different categories: *design, requirement, management, construction*, and *testing*. Additionally, they found that *non-functional requirements, functional requirements*, and *management requirements* are the three rationale types used for decision-making.

The work presented here diversifies itself from the described studies by exploring more general decision rationale in mailing lists. This differs from the work of Soliman *et al.* [7], [8] as a novel data source is explored. The works of Xiong *et al.* [27] and Li *et al.* [28] already explore decision rationale in mailing lists. However, due to the limited scope of these works (*Assumptions* and *Requirements*, respectively), the work presented here complements their work by generating a wider view of the rationale used in mailing lists.

2) Automated AK Classification:

Several studies have been performed introducing automatic means of classifying AK. These studies differ from those discussed in the previous paragraph as classification is generally performed and documented with a coarser level of detail. Sharma *et al.* [29] introduce a tool that extracts decision consensus from mailing lists of *The Python Project*, using a manually built ground truth using eleven different consensus types (e.g. *full consensus* or *partial consensus*). The work of Kleebaum *et al.* [30] proposes a tool that also derives decision rationale, however, from issue tracking systems and version control systems. Their ground truth consists of the high-level classes: the issue, decision, alternative, pro-, and con-argument. The work performed by Li *et al.* [31] and its replication study performed by Fu *et al.* [32] both explore means to automatically identify decision and non-decision sentences in mailing lists using natural language processing models. Similarly, Li *et al.* [33] explored automatic identification of assumption and non-assumption sentences in mailing lists using natural language processing. In the work of Bhat *et al.* [34], an automatic classification of design decisions inside issue tracking systems using the ontology of Kruchten *et al.* [1] is proposed.

Although the work presented here does not address automated classification, it is important that it is reflected against studies that

do as automated classification is a natural application for this work's results. The work performed in these studies [31]–[34] classifies AK concepts on a relatively low level of detail as these only identify one specific type of AK (decisions and assumptions) without addressing these in a larger level of detail. Because the goal of this work is to classify decision rationale on a higher level of detail (addressing multiple types of rationale), future automated classification studies can build upon this fine-grained information, allowing them to classify more information than these previous studies. Although the work performed by Sharma *et al.* [29] is able to classify a larger group of rationale types, their model currently only identifies different types of consensus. The work presented here complements their work by attaching meaning to the made decisions. Currently, the work performed by Kleebaum *et al.* [30] seems to be able to classify on the greatest level of detail, also being able to identify pros and cons (supporting vs opposing arguments). However, similar to Sharma *et al.* [29], the data presented here can contribute to an even finer level of classification. Therefore, although this work itself does not propose an automated AK classification model, it does provide directions for future classifiers.

3) Issue Relationships:

As described in section II-B, no other study has yet explored the relationship between architectural emails and architectural issues, for which there is no other work directly related to this one. However, several other studies have been performed that gained insights into other relationships that issues have or into information-sharing behaviours – topics that could be a source of inspiration for this study. Lüders *et al.* [35] explored relationships between issues attempting to automatically classify these. Their work identified 30 different link types (e.g. dependencies or epics) which they were able to group into 5 super types: *General Relation, Duplication, Temporal/Causal, Composition*, and *Workflow*. Licorish *et al.* [36] explore the information-sharing behaviour of prominent developers in open-source systems. They identified 13 different types of information sharing behaviours, including discussion initiation, instructing contributors, reflection, and information sharing. Finally, a small study performed by Rath *et al.* [37] identified relationships between issues and their respective comment section. They identified three conversation patterns: *Monologues, Feedback*, and *Collaboration*, each describing different levels of involvement that other people have in the issues.

These three studies each identified how any issue has been used in different contexts, providing some context for this work. The work presented here complements all of these by addressing how architectural issues relate to architectural emails in the mailing lists of open-source systems, providing a larger foundation for future work in this direction.

III. Methodology

This section introduces the used research methodology. First, the posed research questions are elaborated on in section III-A. Section III-B describes what data collections are used and how these are used. Finally, section III-C elaborates on the used analysis methods. Figure 1 provides a visual overview of the described methodology.

A. Research Questions

The goal of this research is to identify what types of decision rationale are used in mailing lists and what the relationship is between architectural issues. To complete the goal of this research, a number of research questions are defined.

RQ1 *What types of decision rationale are used in mailing lists of open-source software?*

Answering this question provides insights into the first half of the research goal. Prior research found that software developers share different types of decision rationale in different locations like issue tracking systems [7] and Stack Overflow [8]. An answer to this research question gives insights into the specific types of decision rationale types used in mailing lists. Accordingly, it can be determined what decision rationale types are exclusively or predominantly discussed in mailing lists compared to other sources. If different types of decision rationale are shared in mailing lists, compared to issue tracking systems or forums such as Stack Overflow, this data source might provide a new perspective on software architecture decision-making.

RQ2 *What types of decision rationale co-occur with decision types in mailing lists of open-source software?*

This research question appends RQ1 by exploring the contextual relationships between different rationale types in mailing lists. Multiple types of decisions can be made in architectural emails, like choosing a technology, or a system property [1]. To make these decisions, different types of rationale can be used. However, it is yet unknown what types of rationale are commonly used for different types of decisions. Understanding what decision types and rationale types frequently co-occur (and infrequently co-occur), might suggest effective means for extracting rationale concepts from mailing lists.

RQ3 *What is the relationship between architectural emails and architectural issues of open-source software?*

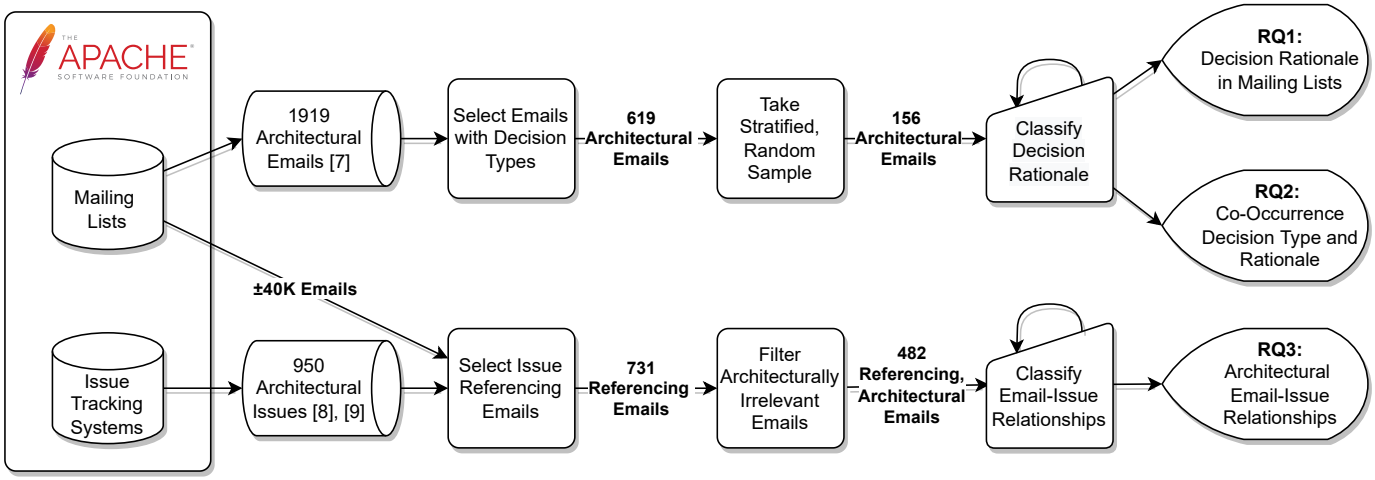

Fig. 1: The methodology used in this work.

Commonly, issues stored in issue tracking systems are referenced inside emails. Issues commonly discuss bugs, feature requests, or contain a description of the current software architecture. Although the literature has explored issue tracking systems in isolation (e.g. [7]), software development commonly uses multiple information sources in parallel, making cross-referencing almost inevitable. Identifying in what context cross-references occur and how these references are done will therefore give deeper insights into the relationship between these different data sources. Understanding these relationships allows developers to see the complete picture, and shows how different sources of information complement each other. Of course, the term *"relationship"* can be interpreted very broadly. In this study, this is specified by exclusively addressing relationships of architectural relevance; i.e. relationships from which AK can be derived.

B. Data Collection

In order to answer the posed research questions, six Apache projects have been selected: Cassandra, Tajo, and the four subprojects of Hadoop: Hadoop-common, HDFS, MapReduce, and Yarn. These projects have been selected as previous research has identified architectural discussions on its issue tracking systems [7], [9], [10] and these projects have publically available mailing lists[1]. The results generated in these studies have been taken as a starting point for the work presented here.

1) Decision Rationale:

To answer RQ1 and RQ2 the dataset built by Lalis [9] is used. Their dataset classified architectural emails of the above-mentioned projects, classifying them using the ontology of Kruchten *et al.* [1] as well as some inductively found classes. In total, their dataset contains 1919 emails spread across 188 mailing threads, of which 619 emails spread across 122 mailing threads contain Kruchten *et al.* [1]'s decision types. From this latter selection, 25 mailing threads containing 153 architectural emails have been randomly sampled to be further analysed and identify architectural decision rationale. Decision types are not equally present in mailing lists, posing a risk to the validity of this work as the sample taken might not well represent the actual data. Therefore, to minimise this risk, a stratified sample is taken that has approximately the same decision type distribution as the complete dataset (see table I for the distribution). Although the work of Lalis includes *Existence Decisions*, *Executive Decisions* and *Bans*, these have been ignored as the first two are represented by their respective subtypes and only a very limited number of *Bans* was detected.

Type	*Total*	*Sample*	Δ
Behavioural	126 (16.89%)	30 (15.79%)	-1.10%
Structural	178 (23.86%)	41 (21.58%)	-2.28%
Technology	146 (19.57%)	34 (17.89%)	-1.68%
Property	76 (10.19%)	24 (12.63%)	+2.44%
Process	220 (29.49%)	61 (32.11%)	+2.61%

TABLE I: Comparison of the decision type distribution in the original dataset [9] and the taken sample.

2) Architectural Email-Issue Relationship:

To answer RQ3 use is made of the data sample generated in the works of Soliman *et al.* [7] and Faroghi [10], who classified architectural issues contained in issue tracking systems using the ontology of Kruchten *et al.* [1]. In total, their dataset contains 950 issues each of which is classified as one or multiple decision types. The architectural issues contained in this dataset have been used to identify the relationship between architectural issues and architectural emails. Currently, there exists no right starting point for identifying architectural email-issue relationships yet. However, it is relatively common for emails contained in the mailing list to reference issues. Therefore, within this study, these references have been taken as a starting point for establishing architectural email-issue relationships.

Although the original intention of this study was to classify entire mailing threads, during the analysis phase, it became apparent that architectural issues are only sparsely discussed throughout mailing lists. To maximise the number of considered architectural issues, and therefore make the found results more generalisable for all architectural issues, without creating significant overhead of to-be-analysed emails, only emails that explicitly reference the architectural issue ID have been considered in this analysis. This includes references like *"CASSANDRA-8844"*, *"C-8844"* and *"C8844"*, however, excludes references such as *"#8844"* or *"8844"* as including these resulted in a substantially larger number of unrelated results. This list of references arose by manually reading emails, adding new reference types to the list as they were found. To determine whether a reference type introduces a lot of irrelevant emails, a sample of the yielded mails was checked whether they actually reference an issue or reference something else (e.g. referencing an error code, line number, or a benchmark result). If the majority of emails had errors like these, the reference type was disregarded in future iterations. Additionally, automatically generated messages (such as pull-request messages generated by Github) are ignored as well, as the information contained in these emails is a literal copy of these other data sources.

To speed up the process of finding such references, an automated approach is used for this process, searching for all issue IDs contained in the dataset of Soliman *et al.* [7] and Faroghi [10][2]. This resulted in a total of 731 emails spread across 416 mailing threads, referencing 284 unique architectural issues. Because this study specifically searches for the relationship between architectural emails (as defined in section II-A) and architectural issues, all emails that reference architectural issues but have no architectural significance are disregarded. Although a definition of architectural emails is used to do this, marking emails as such remains a non-trivial task. Therefore, this is performed in cooperation between researchers, reducing the chance of incorrect classifications.

The used data sample dataset used to answer RQ3 consists of 482 architectural emails spread across 287 mailing threads, referencing 228 unique architectural issues contained in the dataset of Soliman *et al.* [7] and Faroghi [10]. Throughout this work, these emails are referred to as *referencing emails*. The number of referenced architectural issues is much lower than the total number of architectural issues contained in the dataset (950). Firstly, it is likely that not all architectural emails are discussed in mailing lists or architectural emails. Secondly, the automated search used to find references might have simply missed some, as there does not exist a standardised means of referencing architectural issues.

C. Data Analysis

1) Qualitative Analysis:

During the email classification phase for RQ1, the types of decision rationale that are present in mailing lists are identified. To do this, deductive thematic analysis [38] is used, classifying emails using decision rationale types priorly defined in the literature. Classification of emails is performed at a text level, meaning that individual pieces of text are classified as one or multiple rationale types, instead of on an email level (i.e. classifying emails as a whole) as performing this analysis on a sentence level gives a finer-grained understanding of the emails in question. Additionally, this level of granularity opens up more avenues for future work.

[1]All Apache mailing lists can be found at: lists.apache.org

[2]The code can be found at: github.com/wmeijer221/msc_internship

To identify the relationships between architectural emails and architectural issues (RQ3), use is made of inductive thematic analysis [38], as the existing literature has not yet identified such relationships. Differently from the analysis done to answer RQ1, this classification is performed on an email level. This means that every individual email is classified as one or multiple classes. For example, when an email mentions that the referenced issue is part of a *Release Group*, the entire email is classified as such. To do this, the tool built as part of the work of Lalis [9] is used[3]

As qualitative evaluation is inherently subject to bias, a sample of approximately 100 emails was iteratively analysed in cooperation between researchers until a stable set of definitions was created. The identified list of classes was later refined by iterating over the different data samples of the final data set, as suggested in the work of Mayring [38].

2) Quantitative Analysis:

To answer RQ1, the frequency of the found rationale types is shown, giving an indication of what rationale types are prominent in mailing lists (and which are not). This is performed both on a text level (how many pieces of text are marked as a rationale type) and an email level (how many emails have a given rationale type in them). To deepen the acquired insights, the co-occurrence of rationale types is elaborated as well. Instead of doing this on both a text level and an email level, this is only performed on an email level. The reason for this is that quotations commonly only have one type of decision rationale, for which co-occurrences happen infrequently. To answer RQ2, the co-occurrence of decision types [9] and the rationale types found in the same emails are measured. Similarly to RQ1, this is done only at an email level as rationale is commonly widely spread throughout a single email, creating many annotations in the process. Therefore, performing this analysis on a text level would give a skewed perspective on the actual co-occurrence.

To answer RQ3, the frequency of relationship types is presented, giving an indication of how frequently relationships appear in the dataset. As architectural email-issue relationships are classified on an email level, the results are presented in this fashion. The co-occurrence of these relationships is also measured, to gain a deeper understanding of how different email-issue relationships interact with each other.

The results of this work will be statistically elaborated using frequency analysis and co-occurrence. Frequency analysis indicates how frequent classes appear in the analysed dataset, indicating prominence. Co-occurrence is used to identify relationships between classes, indicating how connected they are. In total, three types of tests are performed: 1) co-occurrence of rationale types in emails, 2) co-occurrence of rationale types and decision types in emails and 3) co-occurrence of architectural email-issue relationships in emails. To quantify the co-occurrence of classes, the *Fisher's Exact Test* [39] is used, using a significance threshold of $p < 0.05$. *Fisher's Exact Test* is chosen over other tests (e.g. Chi-Square) to account for the relatively low number of observations in the final dataset as other tests become less accurate with a smaller sample (similar to other studies; e.g. [40] and [41]). This test is applied pair-wise for all possible combinations of classes, such that the used contingency table is a 2×2 matrix containing the number of observations in which the compared classes are present versus when they are not. Per pair-wise comparison that is performed, three hypotheses are defined:

$H_\emptyset$ There is no relationship between the two classes.
H_1 There is a positive relationship between the two classes.
H_2 There is a negative relationship between the two classes.

Where a positive relationship means that the two classes do frequently co-occur (they attract), while a negative relationship means they do not frequently co-occur (they repel). Here, $H_\emptyset$ can be refuted with the calculated two-tailed p-value. H_1 and H_2 can be refuted using the calculated one-tailed p-values in either direction. Here, *"either direction"* means the probability of an arbitrary contingency table having a greater separation between the two tested classes versus an arbitrary table having a smaller separation between the two tested classes; i.e. the direction of the one-tailed test.

To illustrate how Fisher's Exact Test is applied, the relationship between *Assumptions* and *Benefits and Drawbacks* is taken as an example. This relationship can be visualised using the following contingency matrix, where each cell shows the number of emails (not) containing *Benefits and Drawbacks* (*B&D*) and *Assumptions*:

	B&D	*No B&D*
No Assumption	72	38
Assumption	41	2

Applying Fisher's Exact Test on this table, yields a two-tailed p-value of 0.00007, which is well below 0.05 for which $H_\emptyset$ can be refused and the direction of the relationship is tested. The one-tailed p-values returned by the test are 0.00004 and 1, respectively the probability of a relationship being positive (attracting) or negative (repelling). The former is well below 0.05 whereas the latter is not below 0.05 for which H_2 can be refused and H_1 cannot, suggesting that the relationship is attractive.

IV. Results

As explained in section III, this research has two distinct goals: 1) identify decision rationale, and 2) identify relationships between architectural issues and architectural emails. Therefore, the results[4] shown in this section are displayed in a similar fashion, first, by elaborating the results found on decision rationale (RQ1 and RQ2) in section IV-A and then by describing the results found on the relationships (RQ3) in section IV-B.

A. Decision Rationale

The findings of the deductive thematic analysis consist of 557 quotations classified into nine classes: *Solution Benefits and Drawbacks, Constraints, Assumptions, Quality Issues, Solution Risks, Solution Trade-off, Solution Comparison, Decision Rule*, and *Solution Evaluation*. A number of times rationale was found that did not fit any of these classes, for which they are marked *Other*. Section IV-A1 introduces the various definitions of these classes, accompanied by various examples. Section IV-A2 describes the quantitative results.

1) Qualitative Results:

The following section describes the qualitative results generated to answer RQ1. It should be noted that the level of detail used to elaborate on the various classes is strongly influenced by the frequency with which they occur in the dataset. Therefore, the initial paragraphs of this section are relatively large whereas the elaboration of the latter classes is shorter.

Solution Benefits and Drawbacks are the most commonly found rationale type used in architectural emails which describes the benefits and drawbacks (i.e. strengths and weaknesses) of a solution alternative [21], such as their effect on quality attributes (e.g. performance, stability, simplicity, or future-proofness), the features that are supported, or the opportunities that are created as a consequence of an alternative. Sometimes, but not always, such benefits are described in the form of a list, like, *"The benefits are: 1) ..., 2) ..., 3) ..."*.

[3]The Github repository can be found at:
github.com/ArchitecturalKnowledgeAnalysis/EmailDatasetBrowser

[4]The complete results can be found at:
drive.google.com/drive/folders/16paYYiYv2sa2f6xClcCJRnum0XyeU5st

An example of a benefit that addresses quality attributes is *"allow pluggable new authentication methods for UGI, in modular, manageable and maintainable manner"*, which states an improvement of the *pluggability* of the system in a manner that respects the *manageability* and *maintainability* of the system. A more subtle example of this is *"This means that bugs won't pile up and compound each other"*, which is used as an argument for a release schedule, indicating that it improves how bug-prone the system is (an improvement to the *functional correctness* of the system). An example of functional benefits that a solution alternative offers is *"allow multiple login sessions/contexts and authentication methods to be used in the same Java application/process without conflicts, providing good isolation by getting rid of globals and statics"*, which describes a functional improvement that the solution offers. Finally, an example of a benefit that allows for future change is *"The proposal supports using custom memtable implementations to support development and testing of improved alternatives, but also enables a broader definition of 'memtable' to better support more advanced use cases like persistent memory"*, in which the broadened definition of *"memtable"* creates the opportunity for advanced use-cases of the system.

Similarly, drawbacks can address quality attributes, supported functionalities and opportunities as well (however, negatively). An example of drawbacks affecting quality attributes is *"it doesn't seem to add anything else than tight coupling of components, reducing reuse and making things unnecessarily complicated"*, describing the negative impact a solution alternative will have on the *reusability* and *complexity* of the system. An example of drawbacks regarding opportunities is *"CircleCI doesn't cover everything, and with ci-cassandra there are a few things still to do"*, describing the functionalities that are currently still lacking in the described technologies. An example of drawbacks addressing a reduced opportunity for new features would be *"but pushing this strategy to memtable would prevent many features"*. On top of these types of drawbacks, commonly, the impact a solution has on the user experience is addressed as well. An example of such a scenario is *"We waited for so long that we had some assurance JDK6 was on the outs. Multiple distros also already had bumped their min version to JDK7. This is not true this time around. Bumping the JDK version is hugely impactful on the end user"*, where the negative impact on the user is highlighted.

In this class *"plain"* benefits and drawbacks, statements that explicitly mention a strength or weakness, are used most commonly. However, people also state non-drawbacks (statements in which they refute drawbacks) or point out the lack of benefits that a solution offers as well. An example of a non-drawback is *"I don't think labelling features is going to kill the user <–> developer feedback loop"*, where something that was initially presented as a drawback is refuted. Another example of this is *"I personally don't think the productivity hit of adopting a new build tool will be very noticeable (nothing that you can't catch up in a couple of weeks)"*, describing the limited impact changing the build tool will have on the developer productivity. An example of a lack of benefits is *"It's yet another line onto which to cherry-pick, and I don't see why we need to add this overhead at such an early phase"*, implying that the drawback introduced as part of the discussed solution is not outweighed by the benefits it gives.

Constraints are the second most commonly found rationale type and are defined as a limiting factor that specifies what conditions a new solution must adhere to for it to be considered a viable option [17]–[20]. Within architectural emails, constraints are imposed in a number of manners: 1) by the developer community, 2) through requirements, 3) by previous decisions, 4) through technology adoption, and 5) by the existing system.

An example of constraints imposed by the developer community is *"as soon as we're (collectively) confident in a feature's behavior - at least correctness, if not performance"*, where a decision is bound by the confidence that developers have in the behaviour of the solution. Another example is *"as our community bandwidth is precious and we should focus on very limited mainstream branches to develop, test and deployment"*, indicating that the developer community has limited time to spend on new tasks, for which only a limited number of release branches can be managed at the same time. A variant of this is a constraint imposed by the release schedule. An example of this is *"we don't plan on executing on this until after C* 4.0 releases in order to avoid delaying the release"*, where the scope of a feature is reduced to not delay the Cassandra 4.0 release.

New decisions are commonly constrained by various requirements and previously made decisions as well. An example of a user-imposed requirement is *"While the performance impact of migration (if any) could be neglectable to some users, other users could be very sensitive and wish to roll back if it happens on their production cluster"*, where the user requirements in terms of performance are considered to constrain the benefits and drawbacks of the proposed solution. An example of previous decisions influencing the decision-making process is *"we previously agreed limit features in a minor version, as per the release lifecycle (and I continue to endorse this decision)"*, imposing constraints on changes to Cassandra's release process. Another example of this is *"First, classpath isolation being done at HADOOP-11656, which has been a long-standing request from many downstreams and Hadoop users"*, where the long-standing request of users is used as an argument for including an issue into the Hadoop 3.x release, indicating that the decision-making processes are influenced by user requirements.

In some cases, the rate with which users adopt new releases constrains the decision-making process as well. An example of this is *"I know of at least a few major installations, including ours, who are just now able to finish upgrades to 3.0 in production"*, where dropping support for Cassandra 3 is constrained by the rate with which major users are willing to adopt newer versions. Finally, decision-making is constrained by the architecture that is currently present in the system. An example of this is *"We feel this scheme is too different from Cassandra's current distribution model to be a viable target for incremental development"* where incremental development cannot be used as an approach as the current architecture does not allow for incremental changes.

Assumptions are defined as AK taken for granted or accepted as true without evidence [22]. There is no particular type of AK associated with assumptions, for which it always co-occurs with e.g. benefits, drawbacks, or constraints. Assumptions can be recognised by statements such as *"I'm pretty sure"*, *"I can't help but think"*, or *"it should be"*; i.e. statements that introduce a certain level of uncertainty about whatever follows. It should be noted that this uncertainty is different from the uncertainty that is part of *Risks*, as with *Risks* the posed threat is *certain but not always active* whereas with assumptions it is *uncertain but posed as always active*.

In certain cases, the statement explicitly mentions that an assumption is made, like *"Assuming major versions will not be released every 6 months/1 year, ..."*, which literally states an assumption is made. Another example of this is *"The danger I'm anecdotally seeing is that ..."*, where they explicitly state their point is based on anecdotes.

An example of an assumed benefit is *"It is particularly aimed at large clusters, but as a side-effect should improve the small cluster experience as well"*, where the proposed solution *"should"* benefit small clusters as well. Another example of an assumption is *"I think Maven has turned-off some contributors from those language ecosystems who don't know the JVM"*, where the author *"thinks"*

a technology decision has negatively affected people's involvement in the project (i.e a drawback). Finally, an example of an assumed constraint is *"I don't see people rushing to do it until the layers above are all qualified (HBase, Hive, Spark, ...)"*, where the rate of technology adoption is assumed.

Quality Issues are a type of issue that negatively affect a system quality [12], [13], such as performance or maintainability, worsening the overall quality of a system. Because of this, quality issues can be used as a driving factor and rationale to make changes to the system, as removing a negative quality improves the overall system. The analysed emails predominantly used quality issues as an initial driving factor for change, however, occasionally also referenced past and present quality issues to keep in mind when designing new solutions.

Quality issues are relatively easy to recognise as they point out some flawed aspects of the current system, commonly described using some quality attribute. An example of this is *"A single Cassandra process does not scale well beyond 12 physical cores"*, which comments on the scalability of the system. Another example of this is *"Our most common reducer failure is running out of disk space during sort, and this is caused by imbalanced block allocation"*, which sheds light on the insufficient capacity of the system. However, not all quality issues follow this pattern of explicitly referencing quality attributes. An example of this is *"A lot of code has changed between 2.0 and trunk today. The code has diverged to the point that if you write something for 2.0, merging it forward to 3.0 or after generally means rewriting it"*, which is used as an argument to change the release process of Cassandra. This example points out that the current release branches have diverted to such an extent that code is no longer easy to port to a different version, affecting the *Reusability* of newly built artefacts. Although it does not explicitly mention *Reusability*, it does introduce a problem that directly relates to this quality attribute. Finally, an example of taking quality issues in mind when creating new solution alternatives is *"Once a cluster grows sufficiently large, even with topologically aware locality, you eventually want to avoid the everybody-talks-to-everybody situation of current Cassandra, for network efficiency reasons"*, where the proposed solution should take in mind the current *"everybody-talks-to-everybody"* efficiency issue.

Solution Risks are potential undesirable consequences on the software architecture of a system in light of stated quality attribute requirements [2], [17], [23]–[25]. Therefore, *Solution Risks* can be observed as a special type of drawback, having a potential outcome rather than a certain one. In certain cases *Solution Risks* might look very similar to *Constraints*, however, differentiates from this class as *Constraints* always have a certain effect whereas *Solution Risks* do not. *Solution Risks* can be recognised by phrases like *"there is a chance"*, *"could cause"*, *"it might"*, or *"probability"* followed by some negative effect on the system (e.g. performance issues). Similar to *Assumptions*, *Solution Risks* are sometimes called out explicitly, using phrases like *"there is a risk that ..."*.

An example of a risk affecting system reliability is *"One design concern is that replicas of a key range are not stored on the same physical host, as failure of that host could cause the loss of more than one replica of the data"* where the discussed design introduces the risk of data loss during certain circumstances. *Solution Risks* are not limited to the solution domain, as development processes can also introduce risks, for example, *"By consequence, it might slow down the on-boarding of newcomers which we want to make as smooth as possible"*, where not changing the current build tool imposes the risk to reduce the ease with which new developers can join the community. An example of a risk that looks very similar to a *Constraint* is *"I had a similar concern when we were doing 2.8 and 3.0 in parallel, but the impending possibility of spreading too thin is much worse IMO"*, where they introduce the possibility of running out of resources (developers). Although developer throughput *can* impose constraints on the system, its effect is only introduced as a possibility. Finally, an example of an explicit risk statement is *"I think refactoring APIs as a pure reflection of what the DB is doing today just risks ossifying something that grew up organically and probably isn't going to do us any favors"*.

Solution Trade-offs are a description of two or more system qualities that are affected by the same architectural decision, improving some while degrading others (e.g. reduction of performance to improve security) [2], [17]. In the analysed data, low-level and high-level trade-offs were found. Similar to *Assumptions*, low-level *Trade-offs* generally do not appear alone, as they are generally accompanied by *Benefits and Drawbacks*.

An example of a low-level trade-off is *"Our calculations lead us to believe that in fact the shorter rebuild window more than compensates for the increased probability of multiple failure"*, clearly stating that a calculated trade-off is made between the decreased rebuild window and the increased failure chance. Although trade-offs

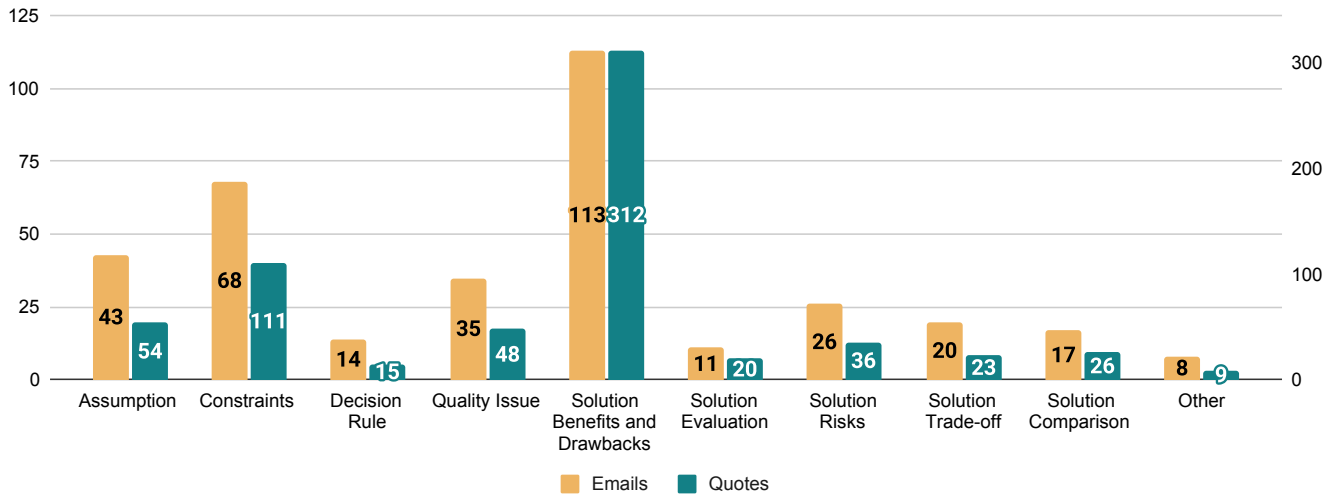

Fig. 2: Frequency distribution of rationale types, emphasising the frequency of rationale types per quote and per email.

must be weighed by the time an architectural decision is made, during the conversation, this is commonly not the case. An example of this is *"Both approaches have pluses and minuses (the usual trade-offs of code-generation vs reflection)"*, introducing trade-offs without stating a specific preference for either direction. *Trade-offs* are not exclusive to new solutions, as they are also made for the development process. An example of this is *"if we keep doing what we've been doing, our choices are to either delay 3.0 further while we finish and stabilize these, or we wait nine months to a year for the next release. Either way, one of our constituencies gets disappointed"*, where a trade-off is made between the two types of user. High-level trade-offs are the least commonly found trade-offs. An example of such a trade-off is *"We prioritized our goals as (1) Reliability (which includes Recoverability and Availability) (2) Scalability (3) Functionality (4) Performance (5) other But then gave higher priority to some features like the append functionality"*, introducing an overall prioritisation of quality attributes as well as an exception to that prioritisation.

Solution Comparisons represent statements in which one or multiple solution alternatives are compared to each other, elaborating their functional or quality differences [21]. Consequently, this leads to solution comparisons commonly being combined with classes such as *Benefits and Drawbacks*, comparing the benefits and drawbacks of different solutions. Because of this, solution comparisons are similar to *Assumptions* as it cannot appear alone – statements always compare *something*. This class should not be confused with *Solution Trade-offs* as solution comparisons require two solution alternatives to be discussed, whereas *Solution Trade-offs* can be made within a single solution. Two different solutions can, of course, make inverse trade-offs, for which such statements can be assigned to both classes.

An example of a solution comparison is *"Has there been consideration given to the idea of a supporting a single token range for a node? While not theoretically as capable as vnodes, it seems to me to be more practical as it would have a significantly lower impact on the codebase and provides a much clearer migration path"*, where a solution using *"virtual nodes"* is compared with *"single token ranges"*, introducing a drawback of the latter in comparison with the first. Beyond being compared for their benefits, solution alternatives can be compared for their compatibilities. An example of this is *"I think 'UDT indexings (at any depth)' can be added because there is no architectural limitation on SAI or SASI"*, where the alternatives are compared in terms of support for a new feature. An example of a comparison between process solutions is *"I suppose in practice all this wouldn't be too different to tick-tock, just with a better state of QA, a higher bar to merge and (perhaps) no fixed release cadence"*, comparing two release strategies. This class does not only compare the *Benefits and Drawbacks* of different solutions, as it compares solutions on a more objective level as well. An example of this is *"this does kind of look like what we tried for tick/tock, but it is not the same"*, comparing the two process solutions of the previous example, without shedding light on their benefits or drawbacks.

Decision Rules are a composite AK concept taking the form of an if-then statement [16]. Decision rules can be subdivided into two components *Condition* and *Consequence*, such that the rule consequence must apply if the rule condition is met. Such rules can be recognised by phrases such as *"if X then Y"*, *"Y unless X"*, or *"Y as long as X"*, where X is a condition and Y a consequence. Decision rules can be used in various manners, like the effect of a decision factor on a quality attribute, possibilities of a solution alternative, or to indicate the limitation of a solution. An example of the first is *"Once MVs reach a point where they're usable in production, we can remove the flag"*, where the usability of a new feature is set as a condition for removing its *"experimental flag"*. An example of a decision rule being used to explore the options of a solution alternative is *"Especially if we go the mono-repo route, then it would make sense to move towards releasing everything together"*, introducing the *"mono-repo route"* as a condition for releasing multiple Cassandra drivers at the same time. An example of the limitations of a solution is *"it should be possible for every major feature that we develop to be a opt in, unless the change is so great and users can balance out the incompatibilities for the new stuff they are getting"*, stating that the proposed solution alternative will work up until *"the change is too great"*.

Solution Evaluations is the least observed class in the dataset and describes statements that give a quantifiable evaluation of a proposed solution or suggest such evaluation is performed [21]. This class is different from *Solution Benefits and Drawbacks* and *Constraints* as a solution evaluation itself does draw a conclusion or constrain decision-making as the evaluation itself does not draw any conclusions from the results. Solution evaluation instances can commonly be recognised by describing or requesting some property of a newly proposed solution, such as their test coverage or complexity.

The majority of the solution evaluation instances address how a solution is tested in some form. An example of this is *"We have been running a QJM-based HA setup on a 100-node test cluster for several weeks with no new issues in quite some time"*, describing a production-scale test. Another example of this is *"given that we've successfully tested rolling upgrade from 2.x to 3.0.0"*, using the successful evaluation of *"rolling upgrades"* to vouch for a proposed solution alternative. A third example of evaluation using tests *"Unit/functional test coverage is pretty high. As a rough measure, there are 2300 lines of test code vs 3300 lines of non-test code"*, describing the unit tests written as part of a new solution. Other solution evaluation instances consist of anticipated evaluations, theoretical evaluations, the maturity of a solution, or a recommendation to perform a solution, though to a very limited extent. An example of the last is *"But this needs to be tested in your cluster to understand the impact"*, recommending to perform a cluster test.

2) Quantitative Results:

The quantitative results gathered to answer RQ1 consist of a total of 153 analysed architectural emails, yielding 557 unique quotations. The frequency distribution (visualised in fig. 2) clearly shows that *Solution Benefits and Drawbacks* are the most commonly found type of rationale, with 312 quotes and 113 emails related to it. This is followed by *Constraints, Assumptions*, and *Quality*

	Assumption	*Constraints*	*Decision Rule*	*Quality Issue*	*Solution Benefits and Drawbacks*	*Solution Evaluation*	*Solution Risks*	*Solution Trade-off*	*Solution Comparison*
Behavioural	7 (11.9%)	14 (15.6%)	3 (15.0%)	9 (18.8%)	26 (17.8%)	5 (31.3%)	9 (23.7%)	6 (22.2%)	4 (21.1%)
Structural	12 (20.3%)	20 (22.2%)	3 (15.0%)	11 (22.9%)	30 (20.5%)	5 (31.3%)	9 (23.7%)	5 (18.5%)	1 (5.3%)
Process	21 (35.6%)	33 (36.7%)	8 (40.0%)	14 (29.2%)	46 (31.5%)	0 (0.0%)	12 (31.6%)	7 (25.9%)	7 (36.8%)
Property	9 (15.3%)	13 (14.4%)	3 (15.0%)	6 (12.5%)	17 (11.6%)	3 (18.8%)	7 (18.4%)	5 (18.5%)	0 (0.0%)
Technology	10 (16.9%)	10 (11.1%)	3 (15.0%)	8 (16.7%)	27 (18.5%)	3 (18.8%)	1 (2.6%)	4 (14.8%)	7 (36.8%)

TABLE II: Frequency distribution of rationale types emphasising the distribution of rationale types across decision types.

Issues, each being present in over 30 emails. Finally, *Solution Risks and Non-Risks, Solution Trade-offs, Solution Comparisons* and *Decision Rules* occurred less than 30 times. Observing these types of rationale, it can be seen that almost all of them are used in emails of any given decision type (visualised in table II). The only exceptions to this are *Solution Evaluation* which is not present in *Process* related emails, and *Solution Comparison* which is not present in *Property* related emails. To gain a deeper understanding of the relationships between rationale types, the co-occurrence of rationale types in emails is analysed. For seven of the explored relationships, the calculated two-tailed *p*-values yielded a significant result ($p < 0.05$) for which the null hypothesis could be rejected. For all of these, using the one-tailed *p*-values, hypothesis two could be rejected, meaning that the established relationship was of a positive nature and these rationale types tend to co-occur.

To answer RQ2, the results generated to answer RQ1 have been used. In three of the performed comparisons (shown in table IV), the established relationships were significant ($p < 0.05$), for which the null hypothesis could be refused. Using the respective one-tailed *p*-values, hypothesis 1 could be refused for each established relationship, suggesting that the established relationship is of a negative nature, suggesting they repel.

Rationale Type	*Rationale Type*	*p*	*Dir.*
Assumption	Benefits and Drawbacks	< 0.0001	$+$
Constraints	Decision Rules	0.0098	$+$
Constraints	Solution Risks	0.0290	$+$
Solution Trade-offs	Benefits and Drawbacks	0.0262	$+$
Solution Trade-offs	Solution Risks	0.0003	$+$
Quality Issue	Solution Risks	0.0186	$+$

TABLE III: Established significant relationships between *Rationale Types*, accompanied with their *Two-Tailed Fisher's Exact Test p*-values and the *Direction* of their relationship: positive ($+$) or negative ($-$). Only positive relationships were identified.

Decision Type	*Rationale Type*	*p*	*Dir.*
Process	Solution Evaluations	0.0034	$-$
Structural	Solution Comparisons	0.0428	$-$
Technology	Solution Risks	0.0169	$-$

TABLE IV: Established significant relationships between *Decision Type* and *Rationale Types*, accompanied with their *Two-Tailed Fisher's Exact Test p-value* and the *Direction* of their relationship: positive ($+$) or negative ($-$). Only negative relationships were identified.

B. Architectural Email-Issue Relationships

The findings of the inductive thematic analysis consist of 482 emails classified into three superclasses: *Issue Group, Issue Reference*, and *Issue Elaboration*. Of these *Issue Reference* is further split into three subclasses *Resource, Discussion Venue,* and *Issue Impact*, and *Issue Group* into three subclasses *Release Group, Feature Group* and *Quality Group*.

In cases where the observed relationship types did not fit any of the created classes, emails have been marked as *Other*. Emails classified as such usually provide too little context for them to be classified accurately, requiring some auxiliary data source for this to be done. Although these emails are not immediately classifiable, they are not excluded from the study as they do contain AK in them and future studies might benefit from including these emails. Alternatively, architectural issues were referenced as an example, as a feature that is back-ported or as a potential source of a new quality issue. However, none of these occurred frequent enough to warrant creating an independent class.

1) Qualitative Results – Superclasses:

Issue Elaboration is the most commonly found class in the data set and is defined as any architectural email that provides insights into the problem or solution domain of the referenced architectural issue. As the name implies, this class is quite broad as architectural issues can be elaborated in a great number of manners. Emails classified as *Issue Elaboration* commonly describe the tackled problem and its importance, rationale for or against solution alternatives or a description of the solution (e.g. impact analysis, component behaviour, or how the described solution is tested). Although intuitive to assume, such emails are not necessarily a lengthy exposition of the ins and outs of the solution alternatives. Many emails simply mention the goal of an architectural issue as a one-liner or share a critique on the discussed solution. An example of this is the phrase *"HADOOP-14556 does it fairly well, supporting session and role tokens"*. An example of the latter is *"In my opinion, this feature of short circuit reads (HDFS-347 or HDFS-2246) is not a desirable feature for HDFS. We should be working towards removing this feature instead of enhancing it and making it popular"*. Although not exclusively, some architectural emails do elaborate on architectural issues in great detail. An example of this is the root email of *"CASSANDRA-10993 Approaches"*, which does the following: 1) it states the goal of the architectural issue *"I'd like to talk about CASSANDRA-10993, the first step in the 'thread per core' work"*, 2) it introduces two solution alternatives *"The first approach models each request as a state machine"* and *"The second approach utilizes RxJava and the Observable pattern"*, and 3) it describes benefits and drawbacks of the two solutions *"The state machines are very explicit (an upside), but also very verbose and somewhat disjointed"*.

Issue Group is the second most commonly found superclass used when referenced architectural issues are put in groups that address a similar topic. This class is subdivided into *Release Group, Feature Group,* and *Quality Group* (elaborated in section IV-B2). These groups are commonly made to create an overview of issues that will be or are tackled together and in some cases affect each other. Non-architectural issues are commonly included in these groups too. An example of this is *"Proposed 11.3"* in which several architectural issues (and non-architectural issues) are grouped as part of a specific release, indicated by the phrase *"I'd like to propose a 11.3 release which is basically a pre-12 with bug fixes added. We are in the middle of testing it and would like to make official tomorrow, if tonight's testing goes well. Here is the list:"*, which is followed by a list of issues, indicating that all those issues *should* be part of version 11.3.

Issue Reference is a class that is assigned to emails that reference information that is contained inside architectural issues, providing context for the discussion inside the mailing thread itself. This class is subdivided into three subclasses *Discussion Venue, Issue Impact*, and *Resource* (elaborated in section IV-B3). An example of this is the email *"[DISCUSS] Tracing in the Hadoop ecosystem"* containing the phrase *"There is a healthy discussion going on over in HADOOP-15566 on tracing in the Hadoop ecosystem. It would sit better on a mailing list than in comments up on JIRA so here's an attempt at porting the chat here"*. Here, the discussion in *"HADOOP-15566"* is referenced to provide context for the discussion that is to follow inside the mailing list. Although in some scenarios the rest of the email elaborates on the referenced issue, this is not always the case as these references are also made to circumvent having to describe the same thing twice (i.e. describing it inside the architectural issue as well as in the mailing list). An example of this part of the conversation in *"row tombstones as a separate sstable citizen"* where one participant says *"Have you taken a look at the new stuff introduced by CASSANDRA-7019? I think it may go a ways to reducing the need for something complicated like*

this", referencing *"CASSANDRA-7019"* as a potential solution to the problem discussed in the email thread.

2) Qualitative Results – Issue Group Subclasses:

Release Group is the most commonly found subclass of *Issue Group*, and is defined as an email in which a decision or a recommendation is given to include or exclude an architectural issue in a specific release (e.g. *"Cassandra 4.0"* or *"the next release"*). These emails commonly describe the status of the grouped issues, their importance, dependencies and their impact. An example of this is the thread *"2.7.3 release plan"*, in which contributors debate why *"HDFS-8791"* should (not) be included in Hadoop 2.7.3, shedding light on the benefits and drawbacks in the meantime, as well as the development processes of Hadoop. An exemplary snippet of this discussion is *"As I expressed on HDFS-8791, I do not want to include this JIRA in a maintenance release. I've only seen it crop up on a handful of our customer's clusters, and large users like Twitter and Yahoo that seem to be more affected are also the most able to patch this change in themselves"*.

Emails classified as a *Release Group* generally take one of two forms: 1) an initial proposal in which a (large) list of issues is grouped and 2) a reply to the root email where a new issue ID or its respective URL is shared with the intention of it being included/excluded in a release. An example of an initial proposal is the root email *"[DISCUSS] Looking to Apache Hadoop 3.1 release"*, in which a group is created using the phrase *"Following is a list of features on my radar which could be candidates for a 3.1 release:"*, which is followed by a list of candidate issues, including multiple architectural issues. One of the replies to this email is an example of the second type, a brief recommendation to include an architectural issue: *"One YARN feature I'd like to add to 3.1.0 is YARN Oversubscription (YARN-1011)"*. Whereas a reply generally provides very little insights, emails with the initial proposal commonly explain the status of given issues (e.g. whether they are completed, have open tasks, or have been assigned to a contributor).

Feature Group is a subclass of *Issue Group* assigned to emails in which issues are grouped because of similarities in the features they address. When architectural issues are grouped for this reason, it is generally done for three reasons: 1) the issues are subtasks of a greater feature, 2) the issues are alternative solutions to the same problem, or 3) the issues address similar features without directly affecting each other. An example of the first is the root email of *"[YARN-2928] first drop on trunk"* describing the status of some subtasks of *"YARN-2928"*. This email includes the phrase *"I think the theme is essentially a basic but complete end-to-end flow that includes the write path and the read path and some UI. These are the key major things we may want to complete before we consider merging the first milestone:"* which is followed by a list of issues including the architectural issue *"YARN-3816"*. An example of the second is the root email of *"[DISCUSS] Hadoop RPC encryption performance improvements"* where *"HADOOP-10768"* and *"HADOOP-13836"* are described as alternative solutions to a specific performance issue: *"There have been some attempts to address this, most notably, HADOOP-10768 (Optimize Hadoop RPC encryption performance) and HADOOP-13836 (Securing Hadoop RPC using SSL)"*. An example of the third is the root email of *"[VOTE] Merge Resource Types (YARN-3926) to branch-3.0"* with the statement *"In summary, resource types adds the ability to declaratively configure new resource types in addition to CPU and memory and request them when submitting resource requests. The resource-types branch currently represents 32 patches from trunk drawn from the resource types umbrella JIRAs: YARN-3926 and YARN-7069"*. This email groups and elaborates the architectural email *"YARN-3926"* with *"YARN-7069"* as these both address resource types.

Quality Group is the third and smallest subclass of *Issue Group* in which issues are grouped to address some quality attribute (like performance or security). An example of this is the root email of *"Performance tickets"* which starts with the phrase *"I'd like to spend some effort in 2.1 improving our performance story for non-io-bound workloads. Here are some of the ideas we have floating around:"* followed by a list of issues, including three architectural issues. These issues are specifically grouped to generate an overview of issues related to performance, in an attempt to improve it. Another example is the mailing thread *"Code quality, principles and rules"*, in which the coding principles used within the Cassandra project are discussed. Among others, the quote *"I agree with the suggestion that it's time to revisit CASSANDRA-7837 and CASSANDRA-10283"* groups the architectural issue *"CASSANDRA-7837"* and another issue *"CASSANDRA-10283"* as part of an endeavour to address these coding practices.

3) Qualitative Results – Issue Reference Subclasses:

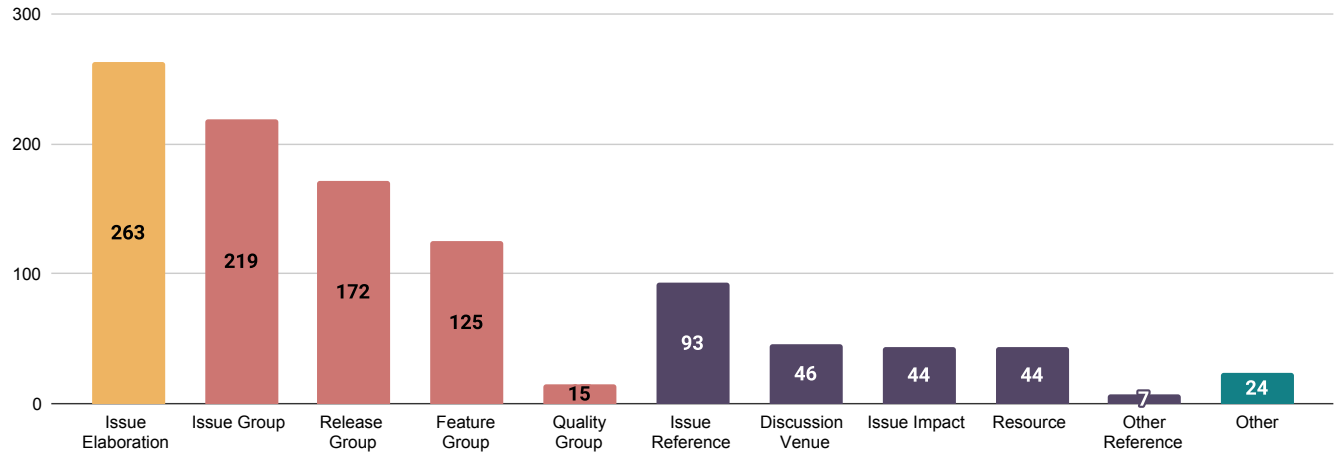

Fig. 3: Frequency distribution of email-issue relationship types per email, including the parent classes *Issue Group* and *Issue Reference*. Frequencies of parent classes are not the sum of the child classes as emails can be classified as multiple relationship types.

Discussion Venue is the most commonly found subclass of *Issue Reference* and is used in mailing lists to reference a discussion that has previously occurred (or is currently occurring) on the referenced issue. In multiple cases, information that was previously shared in the discussion of the architectural issue (e.g. raised concerns, feedback on the proposed solution, or the description of a solution alternative) are referenced and then further elaborated, summarised, or repeated in the referencing email. An example of this is a reply contained in the *"[DISCUSS] Docker build process"* mailing thread, where the email author writes *"As shown from my comments on YARN-7129, I have particular concerns that resonate other posters on this thread"*, referencing the feedback previously given on the issue itself, followed by an elaboration. An example of referencing a previously discussed solution alternative is used in the *"Design for security in Hadoop"* email thread, containing *"Have you looked at HADOOP-4359? In that JIRA, we discussed the idea of using public-key signed capabilities and dismissed it in favor of symmetric-key based capabilities"*.

In other cases, the discussion had in the architectural issue is simply referenced to make a new point, like a design recommendation. An example of this is the *"[VOTE] Merge HDFS-3077 (QuorumJournalManager) branch to trunk"* mailing thread, where the discussion in *"HDFS-3077"* is referenced as the rationale for postponing voting to merge the branch: *"As I indicated in my comments on the jira, I think some of the design discussions and further simplification of design should happen before the merge. See [link to comment on issue]"*. Finally, in some cases, architectural emails attempt to move the discussion location to or from the issue to the mailing list, of which the root email of *"[DISCUSS] Tracing in the Hadoop ecosystem"* is exemplary, containing the phrase *"There is a healthy discussion going on over in HADOOP-15566 on tracing in the Hadoop ecosystem. It would sit better on a mailing list than in comments up on JIRA so here's an attempt at porting the chat here"*.

Issue Impact is the second most commonly found subclass of *Issue Reference* and is used in mailing lists to address the influence that the referenced architectural issue has on new design solutions. Most commonly, the positive impact of an architectural issue is described, as it allows new features to be implemented, or can be used as a solution for new problems. An example of this is the root email of *"[VOTE] Merge YARN-3926 (resource profile) to trunk"*, stating *"Briefly, YARN-3926 can extend resource model of YARN to support resource types other than CPU and memory, so it will be a cornerstone of features like"*, followed by a list of features it unlocks. An example of architectural issues being a solution to new issues is a reply in the *"row tombstones as a separate sstable citizen"* mailing thread: *"Have you taken a look at the new stuff introduced by CASSANDRA-7019? I think it may go a ways to reducing the need for something complicated like this"*, referencing *"CASSANDRA-7019"* as a potential solution for the discussed problem. Additionally, architectural issues marked as *Issue Impact* are referenced as a leading example of future solutions. An example of this is *"proposed new repository for hadoop/ozone docker images (+update on docker works)"*, stating *"We would like to follow the existing practice which is established in HADOOP-14898"*.

The influence of architectural issues is not always positive as they can impose new constraints or additional requirements as well. An example of the former is a reply on the email thread *"Supporting multiple JDKs"* where the constraints imposed by *"CASSANDRA-9608"* are referenced to evaluate the feasibility of another solution alternative by stating *"Some of our java8 code will not compile under java11 (see CASSANDRA-9608); the symbols have literally been removed (Unsafe.monitorEnter() / Unsafe.monitorExit()). Setting -source to '8' will not help. Thus, we need two compilers for the foreseeable future"*.

Resource is a subclass of *Issue Reference* that addresses architectural emails in which an architectural issue is referenced for the information that is shared in it (e.g. design documents, benchmarks, code, trade-offs, subtasks, or examples). Although this information is not necessarily elaborated in the email, it is used to provide context for the rest of the information discussed. Although these are commonly made to simply share information without replicating it, in some cases, issues are referenced to share fundamental knowledge of understanding the content discussed in the rest of the email. An example of the former is a reply in the *"Consistent vs inconsistent range movements"* thread, stating *"Definitely read CASSANDRA-2434. That's probably the best documentation of this feature"*. Another example of this is *"CASSANDRA-10993 Approaches"*, containing the quote, *"I think I outlined the tradeoffs I see between the roll our own vs use a reactive framework in CASSANDRA-10528"*, pointing to the trade-offs of the discussed solution alternatives. An example of the latter is *"[DISCUSSION]: Future of Hadoop system testing"* in which *"HADOOP-6332"* is referenced as prerequisite knowledge for the rest of the email. In this email, the author states *"As many of you know recent development effort from a number of Hadoop developers brought to the existence new system test framework codename Herriot. If you never heard about it please check HADOOP-6332"*, after which they thoroughly elaborate on its future within Hadoop.

4) Quantitative Results:

In total, the quantitative results generated to answer RQ3 consist of 482 architectural emails. The frequency analysis (visualised in fig. 3) shows that *Issue Elaboration* is the most commonly found class, occurring in 263 emails, followed by the other superclass *Issue Group* which is present in 219 emails. The third superclass, *Issue Reference* is present in 93 emails, however, is not the third most common relationship type. Instead, *Release Group* and *Feature Group*, two subclasses of *Issue Group*, are more commonly present, occurring in 172 and 125 emails, respectively. The third subclass of *Issue Group*, *Quality Group*, is the overall least represented class, with only 15 occurrences. The three subclasses of *Issue Reference*, occur an approximately equal number of times, *Discussion Venue* being present in 46 emails, and *Issue Impact* and *Resource* both being present in 44 emails.

Similar to the approach taken to answer RQ1, to gain a deeper understanding of the relationship between the various classes, pair-wise comparisons are performed, observing what relationship types commonly co-occur in the same architectural emails (visualised in table V). In this analysis, all classes (excluding *"Other"* classes

Relationship Type	*Relationship Type*	*p*	*Dir.*
Discussion Venue	Resource	0.0418	+
Feature Group	Issue Elaboration	< 0.0001	+
Feature Group	Issue Impact	0.0142	−
Feature Group	Resource	0.0363	+
Issue Elaboration	Discussion Venue	0.0096	+
Issue Elaboration	Resource	< 0.0001	+
Issue Elaboration	Issue Reference	< 0.0001	+
Issue Group	Issue Elaboration	0.0253	+
Issue Group	Issue Impact	0.0115	−
Issue Group	Issue Reference	0.0035	−
Release Group	Quality Group	0.0004	+
Release Group	Issue Elaboration	0.0003	−
Release Group	Issue Reference	0.0019	−
Release Group	Resource	0.0171	−

TABLE V: Established significant relationships between *Relationship Types*, accompanied with their *Two-Tailed Fisher's Exact Test p-value* and the *Direction* of their relationship: positive (+) or negative (−).

but including the superclasses) have been pair-wise compared for co-occurrence, ignoring relationships between subclasses and their respective superclasses. In a total of 14 instances, the calculated *p*-value of a relationship was of significant strength ($p < 0.05$), for which the null hypothesis could be rejected. Of these, the one-tailed *p*-values of 8 relationships could be used to reject hypothesis 2, for which these results suggest a positive relationship between the pairs. In 6 cases, the one-tailed *p*-values could be used to reject hypothesis 1, for which these results suggest a negative relationship between the pairs.

V. Discussion

Using the results elaborated in section IV, an initial answer can be given to the posted research questions in section III-A. In this chapter, the results of each of the three research questions are discussed and answered in section V-A, section V-B and section V-C, followed by the implications on practitioners and researchers in section V-D and section V-E.

A. What types of decision rationale are used in mailing lists of open-source software?

To answer this RQ1, 153 architectural emails acquired from the work of Lalis [9] were used, yielding 557 quotations classified into nine types of decision rationale: *Benefits and Drawbacks, Constraints, Assumptions, Quality Issues, Risks and Non-Risks, Solution Trade-off, Solution Comparison, Decision Rule*, and *Solution Evaluation*. Of these, *Solution Benefits and Drawbacks* was the most commonly found type of decision rationale, being present in $1.6\times$ more emails, and having $2.8\times$ as many quotations, compared to the second most common rationale type, *Constraints*. This is a result that points in the same direction as the results of Soliman *et al.* [7] who identified that benefits and drawbacks are a common AK concept in architectural issues. Being present in more emails could suggest that the decision-making in architectural emails are largely based on the benefits and drawbacks of the discussed solution alternatives.

Although it is not surprising that the number of quotations is larger than the number of architectural emails, it should be pointed out that the difference between these for *Benefits and Drawbacks* is not comparable with any of the other results. For *Benefits and Drawbacks*, the number of quotations is $2.8\times$ larger than the number of architectural emails, whereas other classes have at most a $1.6\times$ difference. This suggests that in architectural emails in which *Benefits and Drawbacks* are discussed, a multitude of these is discussed or these are discussed in greater detail – to a larger extent than other classes.

Two other rationale types that occurred in over 25% of the analysed emails were *Constraints* and *Assumptions*, respectively being present in 68 and 40 architectural emails. This could suggest that although *Constraints* are not disregarded, they are not frequently used to drive the direction of a decision. Similarly, although *Assumptions* are present in this many architectural emails, the majority of the emails support their statements to some extent. Finally, the other rationale types were found in fewer than 25% of the analysed architectural emails, suggesting that they are not as prevalent as others.

Interestingly, the results show that all of the relationships between rationale types are attractive. These results suggest that the most commonly made *Assumption* is one about *Benefits and Drawbacks*, and that *Solution Benefits and Drawbacks Solution Risks* are commonly part of *Solution Trade-offs*. Beyond that, emails describing *Constraints* are more likely to also address *Solution Risks* and *Decision Rules*. Finally, *Quality Issues* tend to co-occur with *Solution Risks*.

B. What types of decision rationale co-occur with decision types in mailing lists of open-source software?

To answer RQ2, the results generated to answer RQ1 and the results presented in the work of Lalis [9] have been combined, generating an initial overview of the relationship between decision rationale and the decision types defined by Kruchten *et al.* [1]. The results showed that almost all rationale types occur in all decision types (with the exclusion of *Solution Evaluation* and *Solution Comparison*). After performing a deeper exploration of these co-occurrences, three repelling relationships and no attracting relationships were established. The results suggest that *Process* emails repel *Solution Evaluations*, *Structural* emails repel *Solution Comparisons* and *Technology* emails repel *Solutions Risks*. Although significant, these results must be interpreted with a level of care as both *Solution Evaluation* and *Solution Comparison* are infrequently observed classes. This is not the case for the relationship between *Technology* emails and *Solution Risks*.

C. What is the relationship between architectural emails and architectural issues of open-source software?

The data filter step performed to RQ3 showed that approximately two-thirds of the emails that reference architectural issues are *Architecturally Relevant*. Therefore, although the referenced issues are architecturally relevant, it is not inherently true that referencing emails are architecturally relevant as well. The results generated after further analysing the emails that were deemed architecturally relevant, three superclasses were identified: *Issue Elaboration*, *Issue Group*, and *Issue Reference*, of which *Issue Group* is subdivided into *Release Group, Feature Group*, and *Quality Group*, and *Issue Reference* is subdivided into *Discussion Venue*, *Influence of Architectural Issue*, and *Resource*.

Of these, *Issue Elaboration* was most commonly found, shedding light on details within the problem domain, the solution domain, or other outcomes of a design decision. Because this class is the one most frequently found, it is suggested that mailing lists could indeed be a rich source of architectural information that can complement the information shared in architectural issues. The classes *Release Group* and *Feature Group* elaborate on architectural issues in a different fashion. These classes describe architectural issues on a meta-level, respectively specifying when the issue is included in a release (and when not) and how the issue relates to others based on its overarching goal, like a feature or a quality attribute. Potentially, these groups can extend the relationships between issues that are already established in issue tracking systems (Lüders *et al.* [35] presents an extensive list of such relationships).

In this study, *Issue Reference* was subdivided into three additional subclasses. These classes indicate that architectural emails do not merely expand on the information shared on architectural issues. Instead, it showed that architectural issues are commonly referenced to either quickly share the discussions that were priorly had or to reference AK such as the solution design. Additionally, architectural issues were referenced, indicating the influence that architectural issues have. These take the form of additional requirements or constraints, however, also the opportunities architectural issues provide when creating new solutions.

To further understand the established email-issue relationships, their co-occurrence was explored as well. The analysis performed established 14 relationships, of which 8 are attracting and 6 repelling. Of these, 10 relationships involved *Issue Group* or one of its subclasses. When observing these relationships in detail, some things stand out:

- Although *Issue Group* and *Feature Group* tend to attract *Issue Elaboration*, *Release Group* does not.
- *Issue Group* and *Feature Group* seem to share their repelling relation with *Issue Impact*.

- *Issue Group* and *Release Group* seem to share their repelling relationship with *Issue Reference*.
- Although *Feature Group* attracts *Resource*, *Release Group* attracts it.
- *Release Group* has an attractive relationship with *Quality Group*.

These various contradictions show, that although *Feature Group* and *Release Group* share the same superclass *Issue Group*, there are noticeable differences between the relationships of these subclasses.

These results also show that *Issue Elaboration* has a significant relationship with all other classes except *Quality Group*. What stands out here is that although most of them are attractive, *Issue Elaboration* tends to be repelled by *Feature Groups*. This suggests that regardless of the type of reference used, frequently some elaboration of the referenced issue is given. Finally, The results show that *Discussion Venue* and *Resource* commonly co-occur; i.e. when someone references the discussion in an email, they are more likely to refer to a resource as well.

D. Implications for Practitioners

The results of this work have various implications for practitioners. Understanding the types of rationale and architectural issue references used in architectural emails helps practitioners to differentiate between information sources. As software architecture is discussed in various platforms (e.g. mailing lists, issue tracking systems or source code), understanding what these sources contain can help practitioners navigate through the vast amount of AK contained in open-source systems. An example of this is that although *Solution Benefits and Drawbacks* are commonly discussed in mailing lists, rationale types such as *Solution Trade-offs* or *Solution Comparisons* are less frequently discussed. Therefore, looking for these in mailing lists might be less beneficial compared to looking for this AK in other data sources. Similarly, the identified issue-email relationships suggest that many referencing emails contain some form of elaboration of the architectural issue, making mailing lists a potentially valuable source of information.

By establishing relationships between rationale types and between email-issue relationships, practitioners can further improve their AK searching strategies as these relationships can be used to further sieve the AK contained in mailing lists. An example of this is that searching for *Constraints* in architectural emails increases the chances of finding e.g. *Decision Rules*. Similarly, by establishing repelling relationships between decision types and rationale types, practitioners can better manage their expectations regarding what they can find on mailing lists. An example of this is the reduced likelihood to find solution comparisons in emails discussing structural solutions.

E. Implications for Researchers

The work presented in this work allows future research to continue in an array of different directions. This work presented an exploratory study on the decision rationale used in mailing lists and the relationships between architectural emails and architectural issues. Because this work is of an exploratory nature, a data sample of a relatively limited size has been analysed. Therefore, to strengthen or refute the conclusions drawn in this paper, future work could expand the data sample used, generating results of greater significance. This same statement stands for the work performed by Lalis [9], Soliman *et al.* [7], and Faroghi [10], as their datasets are the foundation of this work, consequently limiting the amount of data that could be processed in this study. This especially holds for the dataset of Faroghi [10] as of all 950 architectural issues only 284 were referenced in architectural emails.

Beyond simply increasing the amount of data to strengthen the conclusions of this work, the results posed here can already be used for new endeavours. Although this work resulted in exploring decision rationale in architectural emails, the original goal was to explore all AK concepts. Therefore, various knowledge concepts, such as those explained in the works of Kruchten *et al.* [1], Zimmermann *et al.* [12], [13], and Soliman *et al.* [16], have been included as part of this work's sample study, indicating other AK (non-rationale) that remains hidden in mailing lists. Such a study could also shed light on the inverse of rationale types such as *Benefits* (i.e. *Non-Benefits*) as decision rationale introduced by contributors is occasionally debunked by others. If a future endeavour were to synthesise information contained in mailing lists (or any other data source), such contradictions might prove helpful. Finally, the results of the architectural email-issue relationships identified a large number of emails elaborating on architectural issues. Future work could explore these specific emails in close comparison to their respective architectural issues, identifying precisely how the knowledge contained in them differs.

Of course, as is already attempted by various studies described in section II-C, various researchers have (successfully) attempted to automate the detection of AK in issue tracking systems and mailing lists. By expanding the dataset generated as part of this study, similar models to those present in the literature could be trained for automated data collection and synthesis. Such synthesis technologies could reuse past AK to facilitate future decision-making processes or to recover AK that is not well-documented. In turn, the established relationships between classes could be used to improve these search engines as e.g. *Solution Risks* seem to co-occur frequently with *Constraints*, for which search engines can search for *Constraints* to find *Solution Risks*.

VI. Threats to Validity

This section describes the threats to the validity of our research and the steps taken to mitigate these threats.

A. Construct Validity

The primary research method used in this work is thematic analysis. This method is inherently affected by researcher bias as they need to interpret text and decide if it should be classified and as what it should be classified. To mitigate this threat, the iterative process suggested by Mayring [38] is followed, iterating multiple times over the same data, involving multiple researchers. Furthermore, before performing the complete study, a sample study is performed during which stable class definitions were created, allowing further analysis to be performed in a more systematic and reliable manner.

Secondly, to answer RQ3, an automated means is used to identify referencing emails, potentially missing references that should be included. To mitigate this threat, other means of referencing issues were later included in this method. However, this is not completely fault-proof as references such as *"8844"* and *"#8844"* yield a lot of unrelated results and are therefore references might have been missed. Although this technically threatens the validity of this work, it is considered to be of a very minor extent, as it seems unlikely that how an issue is referenced will affect why it is referenced (though, this remains an assumption).

B. External Validity

Because this study is of an exploratory nature, the amount of data remained relatively limited, limiting the generalisability of the results. To account for this problem to some extent, the *Fisher's Exact Test* [39] is used instead of other tests. Another element that might affect the conclusion validity of this work is the used mailing lists. As all of the analysed projects are Apache projects, it is hard to say how generalisable the found results are. Additionally, Projects

such as Cassandra are very large, containing many architectural emails, for which the results of smaller projects (such as Tajo) might have been overshadowed.

The method used to answer RQ1 and RQ2 used the dataset generated in the works of Lalis [9]. Instead of analysing their entire data sample, only a subset of this data is analysed, for which the analysed dataset might misrepresent the actual data. To minimise this risk, a stratified sample is taken, having an approximately equal distribution of decision types compared to the original dataset.

VII. Conclusions

The goal of this work was to identify what rationale types are used for architectural decisions and what the relationship is between architectural emails and architectural issues, using emails sampled from six Apache projects: Cassandra, Tajo, and the four sub-projects of Hadoop: Hadoop-common, HDFS, MapReduce, and Yarn. Using deductive thematic analysis [38] on the architectural emails identified by Lalis [9], a total of 153 emails were classified, yielding 557 unique quotations spread across nine different types of decision rationale. When exploring these rationale types on a deeper level, seven of these relationships have shown to frequently co-occur. Similarly, by comparing the identified rationale types with the decision types identified by Lalis [9], three repelling relationships were discovered. These results give a clear insight into the types of rationale used in architectural emails and how these relate to the decision types of Kruchten *et al.* [1]. Furthermore, the relationship between architectural emails and architectural issues was explored. To do this, 482 emails sampled from the various mailing lists that reference one of the architectural issues identified by Soliman *et al.* [7] and Faroghi [10] have been analysed. Approximately two-thirds of these emails were considered architecturally relevant and were therefore explored in greater detail. Using inductive thematic analysis [38], seven relationship types were identified, suggesting that architectural emails commonly elaborate on architectural issues, group them, or simply reference them. To further understand these classes, their co-occurrence was measured, yielding eight commonly co-occurring pairs and six repelling pairs. The results generated in this study provide an exploratory overview of decision rationale and architectural issue-email relationships, allowing practitioners to better utilize mailing lists as a source of AK and opening up new avenues for future research.

VIII. Acknowledgements

I would like to thank dr. Mohamed Soliman for his continued guidance throughout this master's internship. Additionally, I would like to thank Andrew Lalis for the development of the email search engine, as well as his feedback and opinions on the various pull requests I submitted during this project.

References

[1] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," en, in *Quality of Software Architectures*, C. Hofmeister, I. Crnkovic, and R. Reussner, Eds., Berlin, Heidelberg: Springer, 2006, pp. 43–58, ISBN: 9783540488200. DOI: 10.1007/11921998_8.

[2] L. Bass, P. Clements, R. Kazman, and a. O. M. C. Safari, *Software Architecture in Practice, 4th Edition*, English. 2021, OCLC: 1247847006, ISBN: 9780136885979.

[3] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," en, *Journal of Systems and Software*, vol. 116, pp. 191–205, Jun. 2016, ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.08.054.

[4] W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin, "How Do Open Source Communities Document Software Architecture: An Exploratory Survey," in *2014 19th International Conference on Engineering of Complex Computer Systems*, Aug. 2014, pp. 136–145. DOI: 10.1109/ICECCS.2014.26.

[5] M. J. de Dieu, P. Liang, and M. Shahin, "How Do Developers Search for Architectural Information? An Industrial Survey," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 58–68. DOI: 10.1109/ICSA53651.2022.00014.

[6] U. A. Mannan, I. Ahmed, C. Jensen, and A. Sarma, "On the relationship between design discussions and design quality: A case study of Apache projects," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 543–555, ISBN: 9781450370431.

[7] M. Soliman, M. Galster, and P. Avgeriou, "An Exploratory Study on Architectural Knowledge in Issue Tracking Systems," en, in *Software Architecture*, S. Biffl, E. Navarro, W. Löwe, M. Sirjani, R. Mirandola, and D. Weyns, Eds., Cham: Springer International Publishing, 2021, pp. 117–133, ISBN: 9783030860448. DOI: 10.1007/978-3-030-86044-8_8.

[8] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch, "Architectural Knowledge for Technology Decisions in Developer Communities: An Exploratory Study with StackOverflow," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2016, pp. 128–133. DOI: 10.1109/WICSA.2016.13.

[9] A. Lalis, "On the Efficacy of Keyword Searches to Find Meaningful Architectural Knowledge in Open-Source Software Mailing Lists," Bachelor's Thesis, University of Groningen, 2022.

[10] S. Faroghi, "Mining Architectural Knowledge in Issue Tracking Systems," Bachelor's Thesis, University of Groningen, 2022.

[11] M. Bhat, K. Shumaiev, U. Hohenstein, A. Biesdorf, and F. Matthes, "The Evolution of Architectural Decision Making as a Key Focus Area of Software Architecture Research: A Semi-Systematic Literature Study," in *2020 IEEE International Conference on Software Architecture (ICSA)*, Mar. 2020, pp. 69–80. DOI: 10.1109/ICSA47634.2020.00015.

[12] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable Architectural Decision Models for Enterprise Application Development," en, in *Software Architectures, Components, and Applications*, S. Overhage, C. A. Szyperski, R. Reussner, and J. A. Stafford, Eds., Berlin, Heidelberg: Springer, 2007,

pp. 15–32, ISBN: 9783540776192. DOI: 10.1007/978-3-540-77619-2_2.

[13] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," en, *Journal of Systems and Software*, SI: Architectural Decisions and Rationale, vol. 82, no. 8, pp. 1249–1267, Aug. 2009, ISSN: 0164-1212. DOI: 10.1016/j.jss.2009.01.039.

[14] J. S. v. d. Ven and J. Bosch, *Agile Software Architecture: Chapter 5. Architecture Decisions: Who, How, and When?* en. Elsevier Inc. Chapters, Nov. 2013, Google-Books-ID: 0kJ1DAAAQBAJ, ISBN: 9780128070253.

[15] H. van Vliet and A. Tang, "Decision making in software architecture," en, *Journal of Systems and Software*, vol. 117, pp. 638–644, Jul. 2016, ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.01.017.

[16] M. Soliman, M. Galster, and M. Riebisch, "Developing an Ontology for Architecture Knowledge from Developer Communities," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Apr. 2017, pp. 89–92. DOI: 10.1109/ICSA.2017.31.

[17] A. Tang and H. van Vliet, "Software Architecture Design Reasoning," en, in *Software Architecture Knowledge Management: Theory and Practice*, M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds., Berlin, Heidelberg: Springer, 2009, pp. 155–174, ISBN: 9783642023743. DOI: 10.1007/978-3-642-02374-3_9.

[18] P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez, "Current Approaches for Solving Over-Constrained Problems," en, *Constraints*, vol. 8, no. 1, pp. 9–39, Jan. 2003, ISSN: 1572-9354. DOI: 10.1023/A:1021902812784.

[19] R. Roeller, P. Lago, and H. van Vliet, "Recovering architectural assumptions," en, *Journal of Systems and Software*, vol. 79, no. 4, pp. 552–573, Apr. 2006, ISSN: 0164-1212. DOI: 10.1016/j.jss.2005.10.017.

[20] M. van den Berg, A. Tang, and R. Farenhorst, "A Constraint-Oriented Approach to Software Architecture Design," in *2009 Ninth International Conference on Quality Software*, ISSN: 2332-662X, Aug. 2009, pp. 396–405. DOI: 10.1109/QSIC.2009.59.

[21] M. Soliman, M. Riebisch, and U. Zdun, "Enriching Architecture Knowledge with Technology Design Decisions," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*, May 2015, pp. 135–144. DOI: 10.1109/WICSA.2015.14.

[22] C. Yang, P. Liang, P. Avgeriou, *et al.*, "An industrial case study on an architectural assumption documentation framework," en, *Journal of Systems and Software*, vol. 134, pp. 190–210, Dec. 2017, ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.09.007.

[23] O. P. N. Slyngstad, R. Conradi, M. A. Babar, V. Clerc, and H. van Vliet, "Risks and Risk Management in Software Architecture Evolution: An Industrial Survey," in *2008 15th Asia-Pacific Software Engineering Conference*, ISSN: 1530-1362, Dec. 2008, pp. 101–108. DOI: 10.1109/APSEC.2008.70.

[24] J. Ropponen and K. Lyytinen, "Components of software development risk: How to address them? A project manager survey," *IEEE Transactions on Software Engineering*, vol. 26, no. 2, pp. 98–112, Feb. 2000, ISSN: 1939-3520. DOI: 10.1109/32.841112.

[25] A. Gemmer, "Risk management: Moving beyond process," *Computer*, vol. 30, no. 5, pp. 33–43, May 1997, ISSN: 1558-0814. DOI: 10.1109/2.589908.

[26] ISO/IEC-25010, "Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models," International Organization for Standardization, Geneva, CH, Standard, 2011.

[27] Z. Xiong, P. Liang, C. Yang, and T. Liu, "Assumptions in OSS Development: An Exploratory Study through the Hibernate Developer Mailing List," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, ISSN: 2640-0715, Dec. 2018, pp. 455–464. DOI: 10.1109/APSEC.2018.00060.

[28] X. Li, P. Liang, and T. Liu, "Decisions and Their Making in OSS Development: An Exploratory Study Using the Hibernate Developer Mailing List," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, ISSN: 2640-0715, Dec. 2019, pp. 323–330. DOI: 10.1109/APSEC48747.2019.00051.

[29] P. N. Sharma, B. T. R. Savarimuthu, and N. Stanger, "Extracting Rationale for Open Source Software Development Decisions — A Study of Python Email Archives," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, ISSN: 1558-1225, May 2021, pp. 1008–1019. DOI: 10.1109/ICSE43902.2021.00095.

[30] A. Kleebaum, B. Paech, J. O. Johanssen, and B. Brügge, "Continuous Rationale Identification in Issue Tracking and Version Control Systems," in *REFSQ Workshops*, 2021.

[31] X. Li, P. Liang, and Z. Li, "Automatic Identification of Decisions from the Hibernate Developer Mailing List," in *Proceedings of the Evaluation and Assessment in Software Engineering*, ser. EASE '20, New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 51–60, ISBN: 9781450377317. DOI: 10.1145/3383219.3383225.

[32] L. Fu, P. Liang, X. Li, and C. Yang, "Will Data Influence the Experiment Results?: A Replication Study of Automatic Identification of Decisions," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, ISSN: 1534-5351, Mar. 2021, pp. 614–617. DOI: 10.1109/SANER50967.2021.00076.

[33] R. Li, P. Liang, C. Yang, G. Digkas, A. Chatzigeorgiou, and Z. Xiong, "Automatic Identification of Assumptions from the Hibernate Developer Mailing List," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, ISSN: 2640-0715, Dec. 2019, pp. 394–401. DOI: 10.1109/APSEC48747.2019.00060.

[34] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "Automatic Extraction of Design Decisions

from Issue Management Systems: A Machine Learning Based Approach," en, in *Software Architecture*, A. Lopes and R. de Lemos, Eds., Cham: Springer International Publishing, 2017, pp. 138–154, ISBN: 9783319658315. DOI: 10.1007/978-3-319-65831-5_10.

[35] C. M. Lüders, A. Bouraffa, and W. Maalej, "Beyond Duplicates: Towards Understanding and Predicting Link Types in Issue Tracking Systems," *arXiv:2204.12893 [cs]*, Apr. 2022, arXiv: 2204.12893. DOI: 10.1145/3524842.3528457.

[36] S. A. Licorish and S. G. MacDonell, "Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study," en, *Information and Software Technology*, Special issue: Human Factors in Software Development, vol. 56, no. 12, pp. 1578–1596, Dec. 2014, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2014.02.004. (visited on 05/27/2022).

[37] M. Rath and P. Mäder, "Request for comments: Conversation patterns in issue tracking systems of open-source projects," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 1414–1417, ISBN: 9781450368667.

[38] P. Mayring, *Qualitative content analysis: theoretical foundation, basic procedures and software solution*, en. Klagenfurt, 2014.

[39] R. A. Fisher, "On the interpretation of χ^2 from contingency tables, and the calculation of p," *Journal of the royal statistical society*, vol. 85, no. 1, pp. 87–94, 1922.

[40] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016, New York, NY, USA: Association for Computing Machinery, Jul. 2016, pp. 165–176, ISBN: 9781450343909. DOI: 10.1145/2931037.2931051.

[41] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, Expectations, and Challenges in Defect Prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1241–1266, Nov. 2020, ISSN: 1939-3520. DOI: 10.1109/TSE.2018.2877678.

APPENDIX

A. Results Example Index

This appendix is added to improve traceability of the examples given in section IV. Per quote contained in a given section, its respective email ID is provided, which can be used in the email browser (github.com/ArchitecturalKnowledgeAnalysis/EmailDatasetBrowser) or in one of the data sheets (drive.google.com/drive/folders/16paYYiYv2sa2f6xClcCJRnum0XyeU5st). Note, in some cases the quotes are slightly altered to improve clarity, increase brevity, fix spelling, or remove information that does not make sense without the broader context of the email. In cases quotes match multiple emails (e.g. the first few quotes in *Assumptions*), an ID is picked at random. The table entries shown here follow the order of reference in the paper as much as possible.

Quoted Text	Email ID
Solution Benefits and Drawbacks	
"The benefits are: 1) ..., 2) ..., 3) ..."	23443
"allow pluggable new authentication methods for UGI, in modular, manageable and maintainable manner"	23443
"This means that bugs won't pile up and compound each other"	24006
"allow multiple login sessions/contexts and authentication methods to be used in the same Java application/process without conflicts, providing good isolation by getting rid of globals and statics"	23443
"The proposal supports using custom memtable implementations to support development and testing of improved alternatives, but also enables a broader definition of 'memtable' to better support more advanced use cases like persistent memory"	41829
"it doesn't seem to add anything else than tight coupling of components, reducing reuse and making things unnecessarily complicated"	41838
"CircleCI doesn't cover everything, and with ci-cassandra there are a few things still to do"	40801
"but pushing this strategy to memtable would prevent many features"	41838
"We waited for so long that we had some assurance JDK6 was on the outs. Multiple distros also already had bumped their min version to JDK7. This is not true this time around. Bumping the JDK version is hugely impactful on the end user"	23403
"I don't think labelling features is going to kill the user <–> developer feedback loop"	32369
"I personally don't think the productivity hit of adopting a new build tool will be very noticeable (nothing that you can't catch up in a couple of weeks)"	43246
"It's yet another line onto which to cherry-pick, and I don't see why we need to add this overhead at such an early phase"	23471
Constraints	
"as soon as we're (collectively) confident in a feature's behavior - at least correctness, if not performance"	32359
"as our community bandwidth is precious and we should focus on very limited mainstream branches to develop, test and deployment"	32832
"we don't plan on executing on this until after C 4.0 releases in order to avoid delaying the release"*	38807
"While the performance impact of migration (if any) could be neglectable to some users, other users could be very sensitive and wish to roll back if it happens on their production cluster"	32832
"we previously agreed limit features in a minor version, as per the release lifecycle (and I continue to endorse this decision)"	40807
"First, classpath isolation being done at HADOOP-11656, which has been a long-standing request from many downstreams and Hadoop users"	23384
"I know of at least a few major installations, including ours, who are just now able to finish upgrades to 3.0 in production"	41039
"We feel this scheme is too different from Cassandra's current distribution model to be a viable target for incremental development"	12779
Assumptions	
"I'm pretty sure"	35062
"I can't help but think"	42551
"it should be"	23469
"Assuming major versions will not be released every 6 months/1 year, ..."	23476
"The danger I'm anecdotally seeing is that ..."	39489
"It is particularly aimed at large clusters, but as a side-effect should improve the small cluster experience as well"	12787
"I think Maven has turned-off some contributors from those language ecosystems who don't know the JVM"	38865
"I don't see people rushing to do it until the layers above are all qualified (HBase, Hive, Spark, ...)"	32792
Quality Issues	
"A single Cassandra process does not scale well beyond 12 physical cores"	15621
"Our most common reducer failure is running out of disk space during sort, and this is caused by imbalanced block allocation"	435
"A lot of code has changed between 2.0 and trunk today. The code has diverged to the point that if you write something for 2.0, merging it forward to 3.0 or after generally means rewriting it"	23748
"Once a cluster grows sufficiently large, even with topologically aware locality, you eventually want to avoid the everybody-talks-to-everybody situation of current Cassandra, for network efficiency reasons"	12787
Solution Risks	
"there is a chance"	32832
"could cause"	12779
"it might"	43241
"probability"	12787
"there is a risk that ..."	42537
"One design concern is that replicas of a key range are not stored on the same physical host, as failure of that host could cause the loss of more than one replica of the data"	12779
"By consequence, it might slow down the on-boarding of newcomers which we want to make as smooth as possible"	43241
"I had a similar concern when we were doing 2.8 and 3.0 in parallel, but the impending possibility of spreading too thin is much worse IMO"	32751
"I think refactoring APIs as a pure reflection of what the DB is doing today just risks ossifying something that grew up organically and probably isn't going to do us any favors"	42551

Quoted Text	Email ID
Solution Trade-offs	
"Our calculations lead us to believe that in fact the shorter rebuild window more than compensates for the increased probability of multiple failure"	12805
"Both approaches have pluses and minuses (the usual trade-offs of code-generation vs reflection)"	2735
"if we keep doing what we've been doing, our choices are to either delay 3.0 further while we finish and stabilize these, or we wait nine months to a year for the next release. Either way, one of our constituencies gets disappointed"	23717
"We prioritized our goals as (1) Reliability (which includes Recoverability and Availability) (2) Scalability (3) Functionality (4) Performance (5) other But then gave higher priority to some features like the append functionality"	424
Solution Comparisons	
"Has there been consideration given to the idea of a supporting a single token range for a node? While not theoretically as capable as vnodes, it seems to me to be more practical as it would have a significantly lower impact on the codebase and provides a much clearer migration path"	12810
"I think 'UDT indexings (at any depth)' can be added because there is no architectural limitation on SAI or SASI"	39730
"I suppose in practice all this wouldn't be too different to tick-tock, just with a better state of QA, a higher bar to merge and (perhaps) no fixed release cadence"	40805
"this does kind of look like what we tried for tick/tock, but it is not the same"	40811
Decision Rules	
"if X then Y"	35062
"Y unless X"	23469
"Y as long as X"	23471
"Once MVs reach a point where they're usable in production, we can remove the flag"	32341
"Especially if we go the mono-repo route, then it would make sense to move towards releasing everything together"	38872
"it should be possible for every major feature that we develop to be a opt in, unless the change is so great and users can balance out the incompatibilities for the new stuff they are getting"	23469
Solution Evaluations	
"We have been running a QJM-based HA setup on a 100-node test cluster for several weeks with no new issues in quite some time"	14300
"given that we've successfully tested rolling upgrade from 2.x to 3.0.0"	32829
"Unit/functional test coverage is pretty high. As a rough measure, there are 2300 lines of test code vs 3300 lines of non-test code"	14300
"But this needs to be tested in your cluster to understand the impact"	28732
Issue Elaboration	
"HADOOP-14556 does it fairly well, supporting session and role tokens"	39035
"In my opinion, this feature of short circuit reads (HDFS-347 or HDFS-2246) is not a desirable feature for HDFS. We should be working towards removing this feature instead of enhancing it and making it popular"	15645
"I'd like to talk about CASSANDRA-10993, the first step in the 'thread per core' work [...] The first approach models each request as a state machine [...] The second approach utilizes RxJava and the Observable pattern [...] The state machines are very explicit (an upside), but also very verbose and somewhat disjointed"	28985
Issue Group	
"I'd like to propose a 11.3 release which is basically a pre-12 with bug fixes added. We are in the middle of testing it and would like to make official tomorrow, if tonight's testing goes well. Here is the list:"	952
Issue Reference	
"There is a healthy discussion going on over in HADOOP-15566 on tracing in the Hadoop ecosystem. It would sit better on a mailing list than in comments up on JIRA so here's an attempt at porting the chat here"	34918
"Have you taken a look at the new stuff introduced by CASSANDRA-7019? I think it may go a ways to reducing the need for something complicated like this"	33504
Release Group	
"As I expressed on HDFS-8791, I do not want to include this JIRA in a maintenance release. I've only seen it crop up on a handful of our customer's clusters, and large users like Twitter and Yahoo that seem to be more affected are also the most able to patch this change in themselves"	27387
"Following is a list of features on my radar which could be candidates for a 3.1 release:"	32146
"One YARN feature I'd like to add to 3.1.0 is YARN Oversubscription (YARN-1011)"	32163
Feature Group	
"I think the theme is essentially a basic but complete end-to-end flow that includes the write path and the read path and some UI. These are the key major things we may want to complete before we consider merging the first milestone:"	25941
"There have been some attempts to address this, most notably, HADOOP-10768 (Optimize Hadoop RPC encryption performance) and HADOOP-13836 (Securing Hadoop RPC using SSL)"	35613
"In summary, resource types adds the ability to declaratively configure new resource types in addition to CPU and memory and request them when submitting resource requests. The resource-types branch currently represents 32 patches from trunk drawn from the resource types umbrella JIRAs: YARN-3926 and YARN-7069"	32650
Quality Group	
"I'd like to spend some effort in 2.1 improving our performance story for non-io-bound workloads. Here are some of the ideas we have floating around:"	20773
"I agree with the suggestion that it's time to revisit CASSANDRA-7837 and CASSANDRA-10283"	30934

Quoted Text	Email ID
Discussion Venue	
"As shown from my comments on YARN-7129, I have particular concerns that resonate other posters on this thread"	36541
"Have you looked at HADOOP-4359? In that JIRA, we discussed the idea of using public-key signed capabilities and dismissed it in favor of symmetric-key based capabilities"	3502
"As I indicated in my comments on the jira, I think some of the design discussions and further simplification of design should happen before the merge. See [link to comment on issue]"	14456
"There is a healthy discussion going on over in HADOOP-15566 on tracing in the Hadoop ecosystem. It would sit better on a mailing list than in comments up on JIRA so here's an attempt at porting the chat here"	34918
Issue Impact	
"Briefly, YARN-3926 can extend resource model of YARN to support resource types other than CPU and memory, so it will be a cornerstone of features like"	31999
"Have you taken a look at the new stuff introduced by CASSANDRA-7019? I think it may go a ways to reducing the need for something complicated like this"	33504
"We would like to follow the existing practice which is established in HADOOP-14898"	36199
"Some of our java8 code will not compile under java11 (see CASSANDRA-9608); the symbols have literally been removed (Unsafe.monitorEnter() / Unsafe.monitorExit()). Setting -source to '8' will not help. Thus, we need two compilers for the foreseeable future"	35090
Resource	
"Definitely read CASSANDRA-2434. That's probably the best documentation of this feature"	30769
"I think I outlined the tradeoffs I see between the roll our own vs use a reactive framework in CASSANDRA-10528"	28986
"As many of you know recent development effort from a number of Hadoop developers brought to the existence new system test framework codename Herriot. If you never heard about it please check HADOOP-6332"	7692