Rijksuniversiteit Groningen

Bachelor's Project Thesis

Exploring Design Decisions in Issue Tracking Systems for Projects in Different Software Domains

Author:
Sarah Druyts

Supervisors:
Dr. M.A.M. Soliman
Prof. Dr. ir. P. Avgeriou

December 12, 2023

Abstract:

Software architecture is a topic of importance for any project which comprises more than two code files. However, formal documentation is often forgotten or deliberately neglected by developers, even though they might have discussed the design elsewhere, such as in an issue tracking system. Afterwards, this informal representation of the design knowledge may be difficult to rediscover.

This thesis investigated potential links between Jira issue characteristics and the issue's design decision content. First, a web GUI for a machine learning (ML) tool (12) was functionally extended to better facilitate manually labeling issues, in order to increase the training dataset for the ML models more easily. Next, using labels predicted by this machine learning tool, statistics were calculated on the patterns between software domains, issue labels, and issue characteristics. Finally, these statistical results were analysed and used to develop heuristics to find issues of certain decision types with higher frequency.

Contents

1 Introduction

Software architecture is a term used to describe various processes and concepts which occur during software development. Concretely, it refers to a high-level structure or model of a system and the components within it. In practice, architecturally relevant elements are often recognised by being the hardest to change once implemented (6). By 2006 (11; 8; 15), researchers were aware that keeping track of the architectural model alone is not sufficient: the reasoning behind certain structures vaporises, and without knowing these intents, new engineers may unknowingly violate undocumented constraints and muddy the design, or worse, create problems or tech-debt down the road. In literature, when this concept of design is separated from the previously mentioned concept of architecture, this collective intent behind a system's architecture is referred to as the **rationale** of the system (1; 6), and the disappearance of rationale knowledge behind decisions is known as **knowledge vaporisation**(11).

Software architecture as a research field has been evolving rapidly over the past few decades. Once initial definitions and concepts were created and accepted (11; 8), we began developing tools to aid us in documentation (4; 7). Now, we have discovered that practitioners do not use these tools sufficiently, so we have turned our focus back towards refining definitions and concepts (6) and finding (better) ways to automate architecture documentation efforts (3; 10).

Design and architecture can be documented in various ways. The most preferred way is to have it formally expressed and collected in one location, however, this is rarely what happens in reality. Nonaka and Takeuchi (14) specified two manners of documentation in addition to this **explicit** level: **tacit**, which is entirely undocumented, and **informally documented**, which is the level this research will focus on. Specifically, to discover this informal knowledge from platforms where developer discussion naturally occurs, such as issue trackers.

As will be detailed in following sections, many efforts have already been made towards facilitating architecture and design documentation, though it is proving to be a difficult problem to solve in its entirety. There can be many aspects behind a developer not adequately documenting their decisions or code, and many different solutions are required to fit all the possible cases. Researchers in this field recognise that, without sufficiently complete documentation, a (large) software system will over time become very costly to maintain or evolve. If one works in practical software development for a long time, one develops an instinct for architecture: which decisions to make to ensure success, and which ones would eventually lead to chaos. Software architecture as a research field aims to formalise this knowledge, generating potential for applications of this information in education and practice.

Since efforts to create tools for developers to use while designing a system seem to not be sufficient, we must find other ways to solve this knowledge gap. As will be detailed in the background section, automatic extraction of knowledge from informal information repositories is still in development, but seems promising. One of the methods used is to apply machine learning tools to the sources of potential informal knowledge, though the main problem here is the lack of sufficiently large datasets. Therefore, this thesis asks the following question: **What are the connections between issue characteristics, architectural knowledge content and software domain of issues in issue tracking systems?**

2 Background

Software is created by making a series of decisions which build upon each other, in order to reach certain stated goals. These **design decisions** can belong to zero or more categories: **Existence**, concerning the presence or absence of (components of) a system, **Property**, stating quality goals such as performance or availability, and **Executive**, documenting dependencies on external code or circumstances (11).

Issue tracking systems, such as Jira, are useful to large teams developing software together. They allow the users to submit **issues** that describe a certain problem or task to be done on the software system. As these issues are in essence a forum for developers to discuss the details of the system and how it should be, it may hold large quantities of design knowledge about the system. The difficulty lies in that this knowledge is informal, and as a result, it is not easy to find, either manually or by using automated methods. Additionally, issues are more than just this forum, being a container for various characteristics assigned to them by the developers. These characteristics can either be inferred directly from the issue, such as the type assigned to the issue, or require preprocessing, such as the length of the description in words or the amount of days between creation and resolution of the issue.

2.1 State of the Art

Alexeeva et al. (1) reviewed the literature on design decision documentation in 2016, stating that the adoption of well-researched methods in practice is not as common as could be hoped. They have provided a list of working points to ameliorate this problem, including more cooperation with the software engineering industry and focusing more on pre-existing systems which may not already have good documentation, as opposed to developing good methods to document design when starting a new project. In the years following, many researchers have taken these points to heart.

That et al. (19) created an approach in 2014 to model architectural decisions in order to let a system automatically check code for violations of these decisions. This may help developers conform to these decisions more, resulting in a more cohesive codebase. Four years later, Schneider et al. (16) created a framework to incorporate informal knowledge into a similar structure referred to as "automated design decision support processes"(16). These methods may help to design good architecture and allow the user to formalise or record informal knowledge, though they are not automated, and may even overly constrain the user, such as in the case of ill-formulated constraints. In the end, methods like this can do nothing for knowledge that has already vaporised out of the consciousness of developers but that may still exist in informal discussion.

In 2018, Buchgeher et al. (3) developed a tool which analyses a project's codebase and generates an architectural model. It is very useful for keeping architectural documentation of a large system with many developers working on it up to date, but it does not take into account the design intents behind the architecture, which the authors specify as future work in their conclusion. Shahbazian et al. (17) created a technique named "RecovAr" in the same year, which connects commits in a repository to issues in an issue tracker in order to find architecturally relevant decisions. The rationale behind this approach is that issues connected to commits which have significantly impacted the architecture will contain the reasoning behind this change. Soliman et al. (18) followed a similar approach, also adding ranking of issues to increase accuracy and analysing the contents of these issues afterwards to create a deeper understanding of architecture in issue tracking systems. These projects analyse the whole codebase and issue tracker of a project, catching many past design decisions. However, some architectural issues may still slip through the net, for example when an issue cannot be linked to an architecturally significant commit, because it did not significantly change the code structure, even though it might contain design knowledge.

Bharadwaj and Kadam (2) used Bidirectional Encoder Representations from Transformers (BERT) natural language processing (NLP) models in order to automatically label GitHub repository issues, for the purpose of facilitating discussion on large projects. In this case, the word label refers to one of *Question*, *Bug* or *Enhancement*. The model has a high accuracy of 0.8653, and though this application was not looking for architecturally relevant issues, this ability to deduce intent from an issue seems promising. Josephs et al. (9) used BERT to create a Slack bot, which records decisions from developer conversations on the messaging platform. One limitation they (and others, such as Keim et al. (10)) encountered was lack of appropriately labeled training data. This approach is promising, though it again lacks the ability to look into the past to find decisions, and as stated, runs into the problem of insufficiently large training set size.

In 2022, Dekker and Maarleveld developed a ML tool to predict the label of an issue based on its characteristics (5), and in 2023, they have refined, expanded and analysed the performance of this tool further (12), including a significant expansion of their training dataset. The expansion of the dataset in this work was partially made possible by the first half of this thesis. For more information on that work, please see the study design section. One problem that still exists with ML tools such as this is that they, by design, function like a black box. More structured and targeted research and results are possible if we are able to find the exact links between issue characteristics and issue labels, which is not something that ML tools of this type can provide. Now, with the results of this work by Dekker and Maarleveld, this type of research is possible.

In conclusion, several methods exist and have been documented in order to aid with design decision documentation. Some of them are proactive, in that they aid the developers while they are creating design decisions, though there is only so much work that can be done in that aspect to mitigate the human error rate of not adequately using provided tools. Therefore, there should be more focus on tools to recover this knowledge from informal sources. One approach to solve this problem is to use ML tools, though there, the problem is the limited size of training datasets. However, now that Dekker and Maarleveld (5; 12) have drastically expanded their training dataset and as a result achieved higher generalisability and precision of their ML models, they were able to generate predictions for a large dataset. Now, it is possible to analyse this generated data and search for connections between issue characteristics and issue labels, that would shed a light onthe actual links being unconsciously used by the black box of ML techniques, allowing research to be more thorough in this topic.

In order to find relevant literature for this section, the following search terms were used on the Springer Link and IEEE Explore databases, in various combinations: *architecture*, *decision*, *design*, *knowledge*, *detection*, *repository*, *machine learning*, *classification* and *issue tracking system*.

3 Study Design

The Background section has outlined the current state of the field of software architecture knowledge (re)discovery and the gaps within it. In order to (re)discover knowledge in a more efficient way, we need more efficient methods. In order to develop these methods, we need more information and data. Therefore, this paper's main research question is: **What are the connections between issue characteristics, architectural knowledge content and software domain of issues in issue tracking systems?** This question can be divided into three subquestions:

- **RQ1**: What types of design decisions are discussed in architectural issues within projects of different domains?
- **RQ2**: How different are the characteristics of architectural issues within projects of different domains?
- **RQ3**: What issue characteristics significantly co-occur with the types of design decisions?

In order to answer these questions, sufficient design decision data about issues is needed. However, manually analysing issues can take a very long time, so instead, this thesis uses the predictions made by machine learning models trained on the existing manually labeled issue dataset. For concerns about the confidence and accuracy of such models, please see the section on threats to validity. There, an analysis of differences between the full dataset and the high-confidence-only set will be presented.

This section will outline the process of facilitating this machine learning model, followed by gathering the results from it and how these are used to then answer this thesis' research questions.

3.1 Project Background

Dekker and Maarleveld (12) have created a commandline tool for machine learning models which is capable of delivering this data. However, the commandline is not the most accessible way of using a tool, and to finetune their models better, a larger dataset was needed. Therefore, part of this thesis project was dedicated to extending ArchUI, a web application interface for their tool which was originally created by the author during a previous project (available with documentation at `https://github.com/mining-design-decisions/maestro-ArchUI`).

Bundled with the CLI, which evolved into an API, this project was named Maestro, and was presented at ECSA 2023 (13). Due to the published version of the tool paper not being available yet at the time of writing, please see Appendix A for the paper itself.

The second half of the thesis project started once the best performing machine learning model had been identified and been used to predict labels for every issue in the database. This data was stored in the database and accessible through the database API provided by Dekker and Maarleveld. For the statistical analysis, it was extracted and processed into a format that was locally stored, to then run all the required tests on.

3.1.1 ArchUI

ArchUI has two types of functionality. The first is to enable the user to use this commandline machine learning tool better by providing graphical web interfaces to create, manage and run machine learning models. Secondly, it allows the user to manually label issues immediately into the model training dataset, and with the potential to add the predictions of the models into the view, so that users can filter and label issues in a more targeted, deliberate way. These two functionalities combine into a snowball effect of labeling issues faster with the help of ML models, and making the models more accurate with a larger dataset.

The ArchUI web application is built in Python, running on Flask for the web server functionality, and uses Bootstrap CSS in Jinja templates to provide a functional user interface. The main page provides information to the user, while the functionalities are grouped into tabs at the top of the page, into the navigation bar. The tool has a login page that connects to the various APIs required for complete function, though some modules can be used without filling in everything. For example, the user can manually label issues without being connected to an active machine learning API.

The tool was built by gathering requirements from the project supervisor and Dekker and Maarleveld, who would be intensely using the tool to help with their own project. The requirements for the application were prioritised in conversation with these stakeholders and consequently, as many of them as were possible were implemented, keeping time constraints in mind.

3.2 Software Domains

The total amount of projects in the database is 1352. The projects with issue counts below 1000 were discarded, because they were less likely to be significant. 238 projects remained, for a total issue count of 1322351. Then, the domain of each project was determined through manual analysis by the author and primary supervisor of this project. Six general software domains emerged from this manual analysis, and each project was assigned one domain. These domains are abbreviated throughout the paper as follows:

- Data Storage & Processing (DSP): Database management, data processing tools.
- Content Management (CM): Storage and processing of higher-level data, documents.
- DevOps And Cloud (DC): Software designed to facilitate DevOps and Cloud efforts.
- SOA And Middlewares (SOAM): Service-Oriented Architecture and software that sits between two other layers, such as database and presentation.
- Software Development Tools (SDT): Such as code editors, debugging tools.
- Web Development (WD): Software focused on web applications, such as server software.

Please find appendix B for a complete list of which project was sorted into which software domain.

It is worth noting that, even though the original dataset contained data from all available open Jira instances, Jira projects that were solely used as bug reporting tools were able to be filtered out through this process.

3.3 High-level Study Design

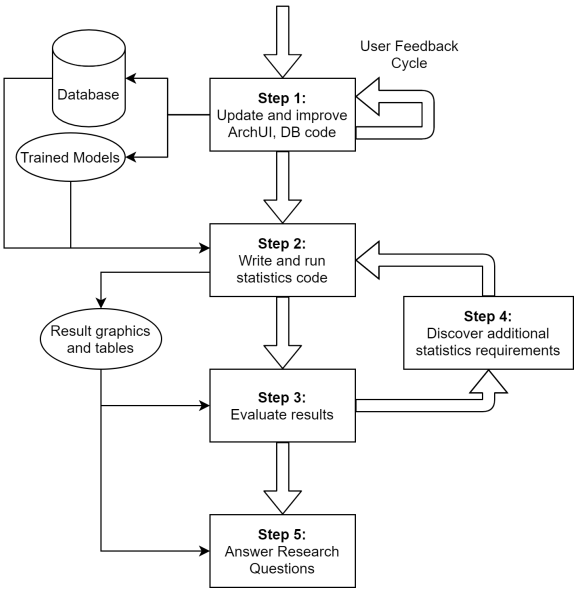


Figure 3.1: The study design process, broken down to high-level steps.

In figure 3.1, the steps undertaken during this project are outlined.

1. **Step 1:** The beginning of the project was to further develop the ArchUI application and assist in the development and testing of the database and ML APIs, as well as to prepare everything needed to generate these statistics, such as assign all relevant Jira projects a general software domain. As stated above, this UI development was done with an active user feedback cycle.
2. **Step 2:** Once all data was ready, it could be statistically tested. Scripts were written to automate this step using Python, matplotlib, and all of this statistical code, along with helper scripts for functionality such as generating the LaTeX tables used in this thesis, can be found at `https://github.com/Shadania/design-decisions-stats`.
3. **Step 3 and 4:** The resulting graphics and tables were great in number, so they had to be evaluated against the research questions: does what we found answer the questions? Are there more things to investigate? With the answers to these questions, a fourth step was added in the process, creating a cycle that was iterated through several times.
4. **Step 5:** Once the results were collected and approved, they could be analysed in detail to answer the posed research questions. In order to get all these results into the thesis

while minimizing human error during repetitive tasks, several scripts have been written to automate certain subtasks, such as converting part of a .csv into LaTeX table code. These scripts can be found in the repository linked above, under the folder "helper_scripts".

3.4 Detailed Study Design

To answer the first research subquestion, the total amount of issues per design decision were counted up per domain, and a Chi-Squared test was run on the resulting counts. In total, the participating projects cumulatively counted 1345784 issues.

For the second and third research subquestions, the data needed to be preprocessed in more detail. This was done using matplotlib in Python, after filtering out non-architectural issues. For this, many characteristics were able to be used from the database API response directly, and for the characteristics where this was not possible or desirable, the calculation methods are below:

- Description size: The amount of words that were not stopwords (from the Python package nltk stopwords).
- Comment count: The amount of comments (≥ 0).
- Comment average size: None if there are no comments, else the average of the non-stopword word count of all comments.
- Hierarchy: Child if the issue is at all a child somewhere, Parent if it is a parent and not a child, and Independent if it is not part of any hierarchy.
- Duration: None if the issue is not yet resolved, else the amount of days between the creation date and the resolution date.
- Resolution: None if the issue is not yet resolved (the status is not either closed, resolved or done), else the value of the field directly from the database issue.

The other characteristics, which were able to be taken over as-is, are:

- Issue Type
- Status
- Amount of Votes
- Amount of Watches

These characteristics are of two types. Hierarchy, issue type, resolution, and status are categorical variables, while the others are continuous variables. For this distinction in type, two different statistical tests were used with the same end goal of answering the research question.

There are two characteristics, the number of attachments and whether or not an issue has a pdf attachment, that were unable to be used, because of difficulties involved in getting an accurate representation of all issue attachments into the source database, with as a consequence that attachment data is not consistent enough to be used for statistical purposes.

3.5 Decision Types

For decision type definitions, this thesis follows the precedent set by the string of previous projects that its author and supervisors were part of. These three categories have already been mentioned in the background section, and are repeated here in order to explain in more detail.

- **Existence**: Issues that describe the (non-)existence of a (component of the) system. For example, to decide that there exists a certain class, and it will have a certain set of functionality.
- **Executive**: Issues describing a change in the external environment of the software system being developed. This can be, for example, a tool change or an external library update. As a rule, decisions that change something about the system's environment but that did not originate from within the system's own requirements, are executive decisions.
- **Property**: Issues that decide something about a quality of the system, such as availability, performance, or security.

3.6 Statistical Analysis: Tests

3.6.1 Chi-Squared Test

For the categorical variables, the chi-squared test was chosen. This test is used to determine if a given set of values for combinations of two variables indicate that the variables are independent or not. The results for chi-squared tests in this thesis are presented in a table of ratios around 1, calculated by dividing the actual counts of issues in this combination by the count expected by the test in the case of variable independence. This means that a value of 1 means that it is as expected (going by the hypothesis of independent variables), a value lower than 1 means it is lower than expected, and a value above 1 means it is higher than expected.

Cells that were left empty did contain independent variables, according to the chi-squared test (p-values of above 0.05). Due to large amounts of small p-values, heavily favoring the alternate hypothesis of these variables not being independent of one another, cells that are significant have been colour-coded: red for ratios of actual/expected that were below 0.8, yellow for between 0.8 and 1.2, and green for above 1.2. This is because the most extreme cases will be the most interesting to examine, and the most actionable to develop, for example, heuristics to focus on finding a certain type of design decision.

Please note that these ratio values, while being the only numerical results of this test presented in this thesis, are not the only results of this test: in the repository mentioned in step 2 of the high-level study design, the full files can be found.

This test is used to answer RQ1 (to measure the independence of the variables domain and decision type), RQ2 and RQ3 (to measure, for every categorical characteristic, the independence of the variables domain and decision type).

3.6.2 Mann-Whitney Test

For the continuous-variable tests, the Mann-Whitney test was chosen. The null hypothesis of this test is that the two given samples were drawn from the same distribution. In this paper, the results of this test have been displayed as arrows drawn on a grouped box plot. An arrow from box plot A to box plot B shows that the distribution indicated by plot A is significantly different from and larger than the one indicated by plot B. This arrow functionality was not available in Python libraries out of the box, so it was coded for this project.

Due to the overabundance of arrows in these results, outliers have been left out of the graphs. Additionally, instead of the usual boundary of significance of 0.05 for p-values, for these tests, a boundary of 0.001 was chosen. Combinations that were below this rate were heavily significant and, as a result, only those are shown. These results have also been displayed in two ways: the primary grouping of the box plot can be either the domain or the decision type, to allow for detailed comparisons between both and yet condense the amount of individual graphs.

As with the chi-squared test, there are more versions of these Mann-Whitney significance graphs in the same linked repository.

This test is used to answer RQ2 and RQ3 (to measure, for every characteristic, which combination of decision type and domain has the highest/lowest significantly different mean from the other combinations).

3.7 Expected Difficulties

3.7.1 Results Size

There will be many results: the final data has several dimensions, with each dimension having several possible values: decision type, issue characteristics, possible values for each characteristic. To display these results in a digestible manner and still show it in a detailed enough way to draw useful conclusions from will be a challenge, which is mitigated by relegating overly crowded plots to the appendix and favouring the simple decision model in the text of the paper itself, and where applicable, the full-domain model instead of showing all domain-specific results. The results for the simple results model are calculated as follows: an issue that has two or more decision type labels counts once for all of its labels. For example, an issue that is both existence and property will count for both existence and property.

3.7.2 Status, Resolution, Issue Type

Additionally, the status, resolution and issue type characteristics brought another difficulty with them in the form of an incredibly large amount of potential values. The values of these characteristics are user-defined, and thus dependent on the Jira instance and project. It is also possible for two projects to use different values for the same purpose, or the same value for different purposes. This made it difficult to narrow down the values to analyse, and to interpret the results keeping these factors in mind. The values for these characteristics that are shown in this thesis were chosen after several iterations, by the following criteria:

1. The value has to be present in three or more domains.

2. The value has a sufficiently high rate of occurrence within each domain, so we can reasonably assume that every domain had the same level of access to it, and was not prevented from using it because it wasn't available in the Jira instance or there was an overly popular alternative value for the same/similar purposes.

Other approaches that were considered and rejected include:

1. Picking the issues with the highest counts across all domains. This was not ideal because some domains have so many issues that have one value while other domains do not have a single one, and this method does not respect that, so it cannot generate usable results for these values.

2. Picking only the Jira defaults with the highest counts. This method also did not take into account that some domains do simply not use certain default values, or not as much as they maybe should, because they have replaced the function of these values with custom values. On the other hand, some custom values are so prevalent across domains that they might as well be default, and this method cannot take that into account.

4 Results

In this section, the results gathered through the process described in the study design section will be shown, as well as interpreted. Each research question is answered in its own subsection.

4.1 RQ1: Design Decisions per Domain

RQ1: What types of design decisions are discussed in architectural issues within projects of different domains?

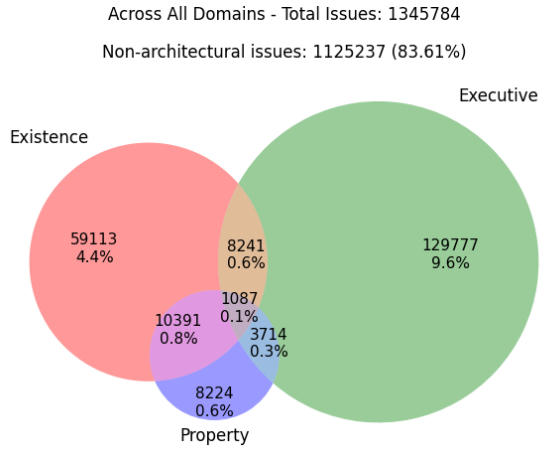


Figure 4.1: A Venn diagram of the amount of issues divided over the design decision types, across all domains.

To answer the first research question, we must count how many issues per decision type there are per domain, and use the chi-squared test to determine if there are significant differences. A total count of issues across all domains, grouped by decision type and visualised as a Venn diagram, can be found in figure 4.1. This same information but separated per domain can be found back in figure 4.2.

A chi-squared test was run on the amounts of issues per decision type, per domain, to examine if there were (in)dependent variables. Table 4.1 shows the results of this test for intersected decision types, and table 4.2 for simplified types. As stated in the study design section, the numbers in these tables signify the rate of issues of this type found divided by issues of this type expected if the variables examined are truly independent.

We can conclude from these results that Content Management, Software Development Tools and Web Development seem overall less rich in design decision content than the average across domains. As the only exception for these three domains, Web Development has a slightly higher-than-average rate for pure **executive** decisions (1.05 in the simplified view, 1.09 in the intersected view).

Data Storage and Processing seems richer than average in **existence**, but especially in **property** (1.44 in the simplified view, whereas in the intersected view 1.44 for the single label, 1.65 combined with **existence**, and 1.13 in all), while being poorer than average in **executive** (0.74 in the simplified view, while in the intersected view 0.72 for the single label and 0.78 combined with **existence**).

DevOps and Cloud is richer than average in most decision types, especially **executive** (2.04 in the simple view, while in the intersected, the single label gives 2.04, combined with **existence** 2.81, combined with **property** 1.31, and all types combined 1.97), but not **property** (0.91 in the simple view, while in the intersected view 0.73 for the single label, and 0.83 combined with **existence**. Values for when **executive** is combined seem to skew higher because **executive** counts are more significantly different and higher than **property** counts are significantly different and lower than average).

SOA and Middlewares seems to be around average, with a slight preference for **executive** decisions (1.10 in the simplified view, 1.12 for the single label in the intersected view, no other significant values) and slightly fewer **property** decisions than average (0.93 in the simplified view, 0.91 for the single label and 0.92 for combined with **existence** in the intersected view).

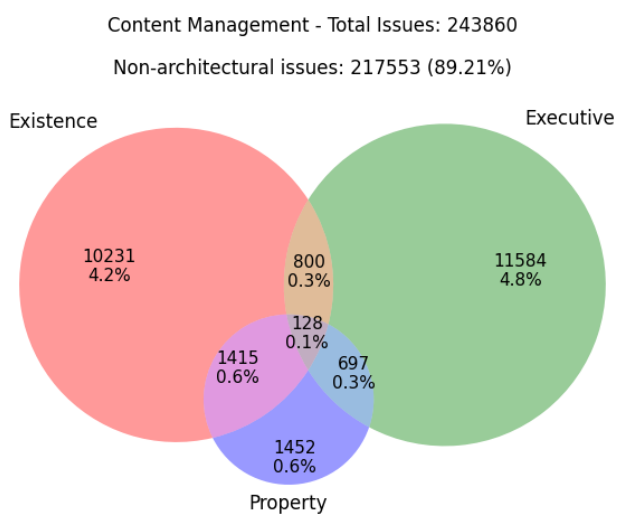
Key Takeaways for RQ1:
The most prevalent decision type is **executive**, and **property** the least. Combinations of all decision types appear across all domains.

The domain DevOps and Cloud seems the most productive to find decisions of type **executive** and **existence**. Data Storage and Processing is a good complement, supplying the **property** decisions.

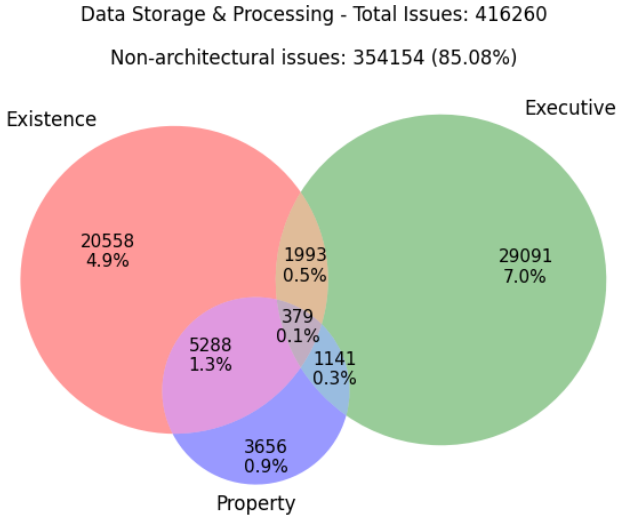
4.2 RQ2: Issue Characteristics per Domain

RQ2: How different are the characteristics of architectural issues within projects of different domains?

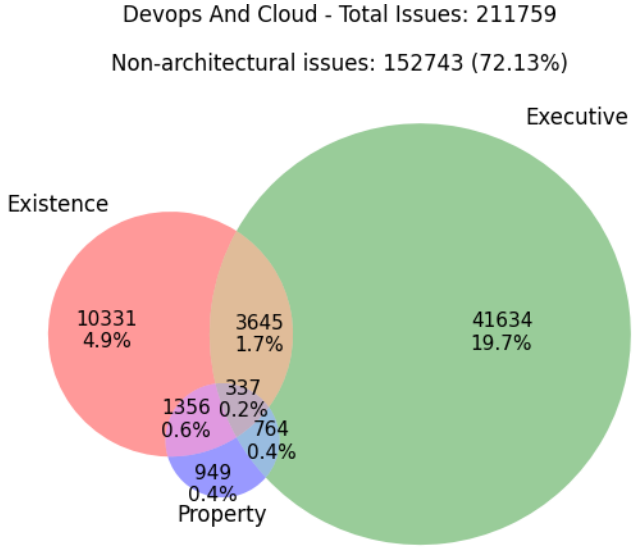
The categorical issue characteristics have been analysed per domain by running a chi-squared test. The results of this are available in tables 4.3



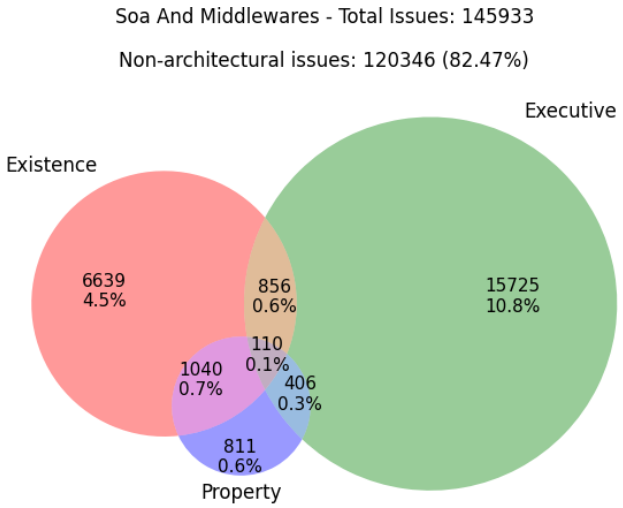
(a) Content Management



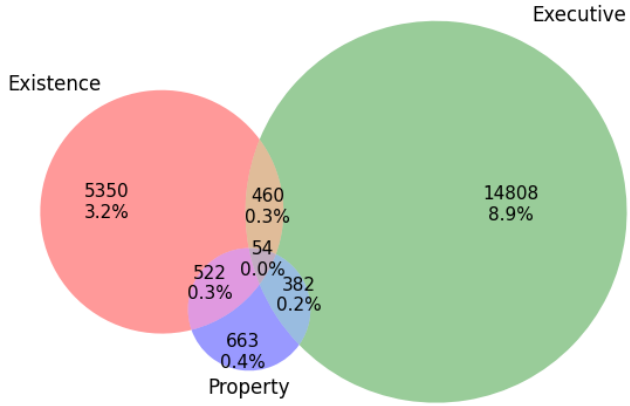
(b) Data Storage & Processing



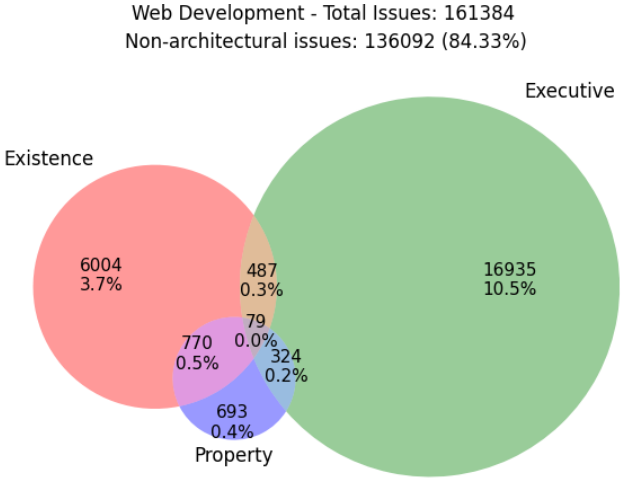
(c) DevOps and Cloud



(d) SOA and Middlewares



(e) Software Development Tools



(f) Web Development

Figure 4.2: Venn diagrams of the amount of issues divided over the design decision types, per domain.

Domain	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
CM	0.96	0.49	0.54		0.75		0.65	1.07
DSP	1.12	0.72	0.78	1.44	1.65		1.13	1.02
DC	1.11	2.04	2.81	0.73	0.83	1.31	1.97	0.86
SOAM	1.04	1.12		0.91	0.92			0.99
SDT	0.73	0.92	0.45	0.65	0.41	0.83	0.40	1.04
WD	0.85	1.09	0.49	0.70	0.62	0.73	0.61	1.01

Table 4.1: Chi-squared test results on decision types per domain, with intersected decision types. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Domain	Exis	Exec	Prop	Non-Arch
CM	0.88	0.51	0.87	1.07
DSP	1.15	0.74	1.44	1.01
DC	1.25	2.04	0.91	0.85
SOAM		1.10	0.93	0.99
SDT	0.66	0.90	0.56	1.05
WD	0.78	1.05	0.67	1.02

Table 4.2: Chi-squared test results on decision types per domain, with simplified decision types. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Domains	Independent	Child	Parent
CM	1.07	0.37	0.44
DSP	0.88	2.04	1.87
DC	1.07	0.33	0.49
SOAM	1.02	0.81	
SDT	1.01		0.82
WD	1.02	0.88	0.76

Table 4.3: Chi-squared test results on the hierarchy characteristic's values, per domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

(hierarchy), 4.4 (issue_type), 4.5 (resolution) and 4.6 (status).

Mann-Whitney tests were run for the continuous issue characteristics, the results of which are available in figure 4.3.

Categorical Variables

From table 4.3 we can infer that domain Data Storage and Processing uses hierarchical issues a lot more than the other domains, especially overshadowing Content Management and DevOps and Cloud.

In table 4.4, the domain with the relatively greatest amount of sub-tasks is data storage and processing. DevOps and Cloud and Software Development Tools have counts of this value for issue type that are not significantly different from the average. All other domains (Content Management, SOA and Middlewares, Web Development) have significantly less than average amounts of this value for issue type. Next, domains with greater than average amounts of bugs are Content Management and Software Development Tools, followed a little less closely by DevOps and Cloud. The column for improvements is an interesting gradient, with Software Development Tools having relatively few and Data Storage and Processing having relatively many. Interestingly, Software Development Tools also has relatively few new features, but relatively the most tasks. Software Development Tools also has relatively many tasks, while Content Management and Data Storage and Processing have relatively few. Wishes are the most relatively prevalent in Content Management and the least in DevOps and Cloud.

For resolution, from table 4.5, we see that Content Management, Data Storage and Processing and DevOps and Cloud have relatively (very) few Done issues, while SOA and Middlewares, Software Development Tools and Web Development have relatively (very) many. For bug-related values won't fix and not-a-bug, Software Development Tools and Content Management have relatively many, while Data Storage and Processing and Web Development have relatively few.

Table 4.6 shows the status characteristic's chosen values across domains. Immediately, we can see that no domain has as relatively many open, reopened and in-progress issues as Data Storage and Processing. Additionally, this domain also has the highest relative amount of resolved issues amongst domains, followed very closely very SOA and Middlewares. This domain seems to heavily favour that value, having less-than-average relative rates for all the other values. DevOps and Cloud largely has smaller-than-average relative rates for open, resolved and reopened, instead

Domains	Sub-Task	Bug	Improvement	New Feature	Wish	Task
CM	0.37	1.50	1.10	0.95	1.61	0.71
DSP	1.38	0.84	1.17		0.93	0.70
DC		1.12	0.78	1.06	0.46	1.14
SOAM	0.58	0.83	0.83	1.14	1.18	1.57
SDT		1.30	0.63	0.72		1.36
WD	0.73				0.77	1.14

Table 4.4: Chi-squared test results on the issue_type characteristic's values, per domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Domains	Won'T Fix	Duplicate	Fixed	Obsolete	Not A Bug	Done
CM	3.05	1.55		2.65	2.56	0.07
DSP	0.96	1.06	1.15	0.07	0.46	0.57
DC	0.37	1.30	1.19	1.13		0.52
SOAM	0.71	0.39	0.95	0.53		1.45
SDT	1.39		0.56	2.49	1.55	2.16
WD	0.91	0.32	0.59	0.61	0.42	2.49

Table 4.5: Chi-squared test results on the resolution characteristic's values, per domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Domains	Closed	Open	Resolved	In Progress	Reopened
CM	1.19	0.86	0.54	0.68	
DSP	0.59	1.92	1.71	1.42	1.81
DC	1.34	0.34	0.34	1.30	0.20
SOAM	0.78	0.79	1.70	0.59	0.55
SDT	1.28	0.71	0.36	0.58	
WD	1.07	0.76	0.93	0.35	

Table 4.6: Chi-squared test results on the status characteristic's values, per domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

having relatively many issues that are closed and in progress.

Continuous Variables

Figure 4.3 shows the results of the Mann-Whitney test on the remaining issue characteristics. Domain Data Storage and Processing has the clear advantage in comment count over the other domains, while competing with Content Management and DevOps and Cloud for the average comment size. Content Management also sticks out above the others in description size, duration and votes. Notably, for votes, the other domains do not seem to use this feature nearly as much. For watches, Data Storage and Processing narrowly has the lead, over Content Management.

Key Takeaways for RQ2:
The domain of Data Storage and Processing again seems to make heavy use of the examined Jira issue features compared to other domains, for example, the issue hierarchy system. The domain Content Management also uses votes and comments, but not hierarchy, implying higher participation from its users but lower cohesion across issues. Additionally, its issues seem to have a significantly longer duration than any other domains. Also, Content Management seems to treat relatively many bugs compared to other domains, reflected in issue type and resolution. Data Storage and Processing seems more relatively active, from status, and architecturally-concerned, from issue type.

4.3 RQ3: Issue Characteristics per Design Decision Type

RQ3: What issue characteristics significantly co-occur with the types of design decisions?

The categorical issue characteristics have been analysed per domain by running a chi-squared test. The results of this are available in tables 4.7 (hierarchy), 4.8 (issue_type), 4.9 (resolution) and 4.10 (status).

It is worth noting that due to the size of the data, the intersected decision type tables, as well as the complete simple decision type tables, have been relegated to appendix C of this report. In this section, only the simplified full-domain decision type model results will be shown.

For the continuous issue characteristics, Mann-Whitney tests were run, the results of which are available in figures 4.6 (average comment size), 4.5 (comment count), 4.4 (description size), 4.7 (duration), 4.8 (votes), and 4.9 (watches).

For the continuous variables as well, only the simplified decision type model is shown within the thesis. The intersected decision type model can be found in appendix C.

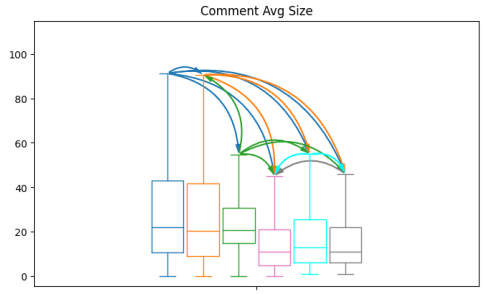
Categorical Variables

All Domains	Exis	Exec	Prop	Non-Arch
Independent	0.93	0.99	0.96	1.01
Child	1.60		1.05	0.96
Parent	3.21	1.89	3.98	0.67

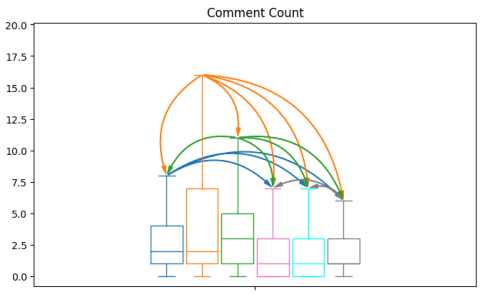
Table 4.7: Chi-squared test results for the relation between the hierarchy characteristic's values and simple decision type contained in the issue, across all domains. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

With the hierarchy characteristic (table 4.7), across all domains we see marked increase in parent issues being architectural. The numbers for independent issues are not very far away from average, but still show that they tend towards non-architectural, except for **executive-property** type issues. Child issues have a more complicated story: they favour **existence** and **existence-property** types, and are less likely to contain any multi-label decisions with **executive**. Notably, there are no significant differences from expected in **executive** and **property** prevalence for child issues. Noteworthy is that in Content Management, the domain which used relatively few hierarchical issues, both child and parent issues have higher rates to contain any kind of architectural decision than the domain average, except child issues are extremely unlikely to contain **executive-property** type decisions. DevOps and Cloud, the other domain which rarely used hierarchical issues, seems to have used them in the opposite manner: its rates for parent and child issues are all lower than the domain average, except for **existence** issues.

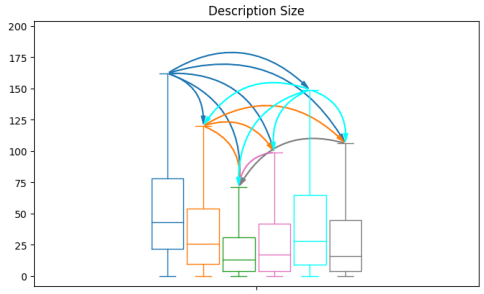
(a) Color legend for this figure.



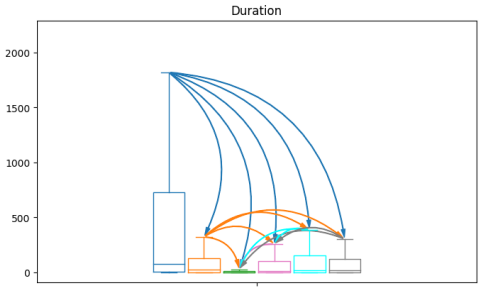
(b) Comment average size.



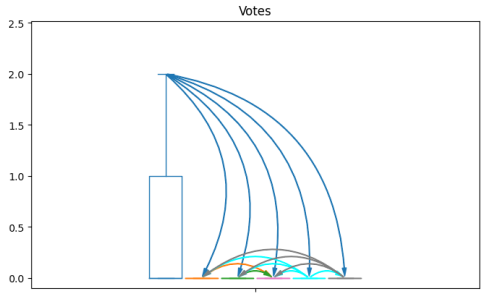
(c) Comment count.



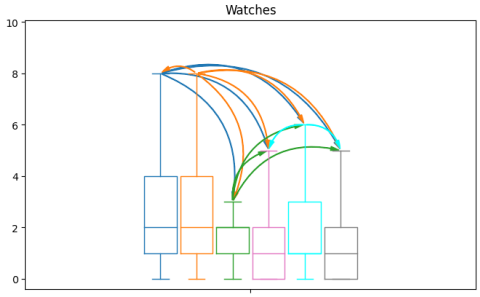
(d) Description size.



(e) Duration.



(f) Votes.



(g) Watches.

Figure 4.3: Box plots of the continuous data issue characteristics, per domain. Arrows indicate significantly larger means according to the Mann-Whitney test.

All Domains	Exis	Exec	Prop	Non-Arch
Sub-Task	1.93	1.33	1.11	0.92
Bug	0.30	0.41	0.37	1.10
Improvement	1.87	1.35	2.42	0.89
New Feature	4.40	2.08	3.32	0.67
Wish	1.90	2.55	2.06	0.79
Task	1.11	3.12	0.88	0.81

Table 4.8: Chi-squared test results for the relation between the issue_type characteristic's values and simple decision type contained in the issue, across all domains. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

In table 4.8, we notice that bugs are significantly different from expected and unlikely to contain any kind of architectural knowledge. On the other hand, improvements, new features and wishes are more likely than average to contain all types of architectural decision. Sub-tasks are also slightly more likely than average to contain all types of architectural decision, with the most extreme being a value of 1.93 for **existence**. Perhaps surprisingly, tasks are slightly more likely than average to contain **existence**, over three times as likely as average to contain **executive**, and slightly *less* likely than average to contain **property**. This last detail is the one exception to the statement that other than bugs, all issue types seem to contain architectural knowledge more often than average.

All Domains	Exis	Exec	Prop	Non-Arch
Won'T Fix	1.33	0.58	1.27	1.03
Duplicate	1.09	0.81	1.21	1.02
Fixed	0.97	1.04	1.02	1.00
Obsolete	0.69	0.51	0.46	1.10
Not A Bug	0.31	0.36	0.44	1.15
Done	0.98	1.18	0.81	0.98

Table 4.9: Chi-squared test results for the relation between the resolution characteristic's values and simple decision type contained in the issue, across all domains. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

From table 4.9, we first notice that not-a-bug and obsolete seem to have the same pattern as issue type bug: significantly not architectural. We also notice the pattern in the **executive** column: fewer than average get assigned as duplicate or won't fix, but more than average get done. **Existence** and **property** show an opposite pattern: significantly many of them get assigned duplicate and won't fix, while fewer get assigned done.

All Domains	Exis	Exec	Prop	Non-Arch
Closed	0.89	1.07	0.80	1.00
Open	1.52	0.75	1.92	0.98
Resolved	1.05	0.91	1.12	1.01
In Progress	2.68	1.11	2.56	0.84
Reopened	1.28	0.66	1.32	1.02

Table 4.10: Chi-squared test results for the relation between the status characteristic's values and simple decision type contained in the issue, across all domains. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

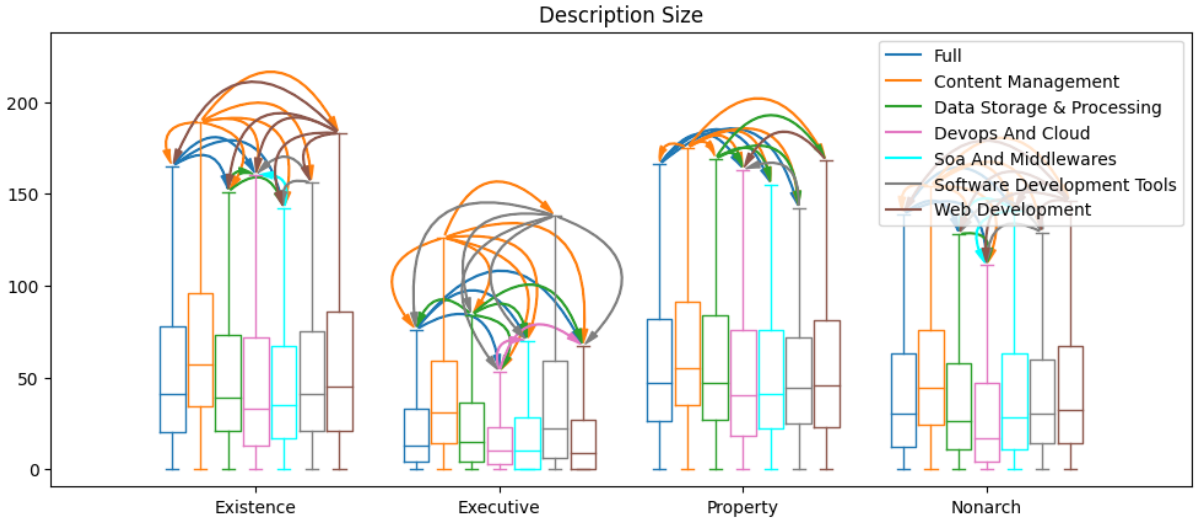
For the last categorical variable, status, we examine table 4.10. We see a similar pattern of **existence** and **property** agreeing on all values, while **executive** goes its own way. **Existence** and **property** are less likely to be closed, instead being more likely to, in ascending order, be resolved, reopened, open or in progress. **Executive** is significantly less likely to be open or reopened, and slightly more likely to be in progress and closed.

Continuous Variables

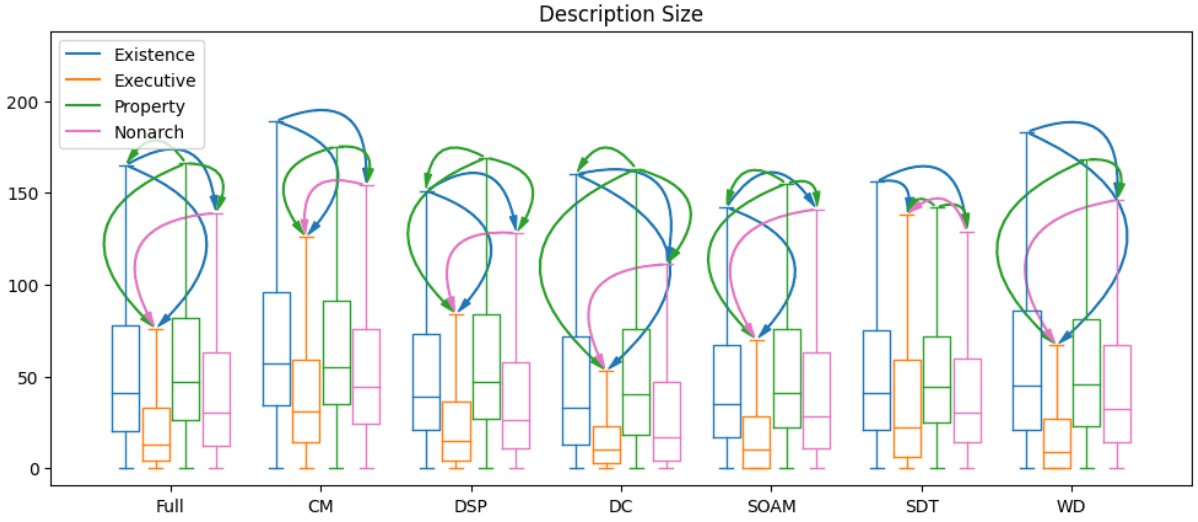
For description size, what immediately jumps out from figure 4.4 is that **executive** decisions have decidedly the smallest descriptions. Next, Content Management has many arrows pointing at other domains, reconfirming what we saw in RQ2, which is that it usually has the largest descriptions. From the two most prevalent arrow colors in the second graph, we can further infer that the triple label issues and **existence-property** issues have similar advantages over other decisions across all domains, in that order.

Data Storage and processing clearly stands out over the other domains in comment count (table 4.5) for all types, except for **executive** decisions. There, DevOps and cloud has the highest mean. On the flip side, **property** decision containing issues seem to have more comments on average, at least when taken across all domains, and especially in content management and data storage and processing.

An interesting observation for average comment size (table 4.6) is that for every decision type, a different domain sticks out to have the largest comments. If we look at the decision-type-first graph, the full-domain collection of box plots indicates that **property** has the longest comments, followed by **existence**, and perhaps surprisingly, **executive** has the shortest comments. It is only

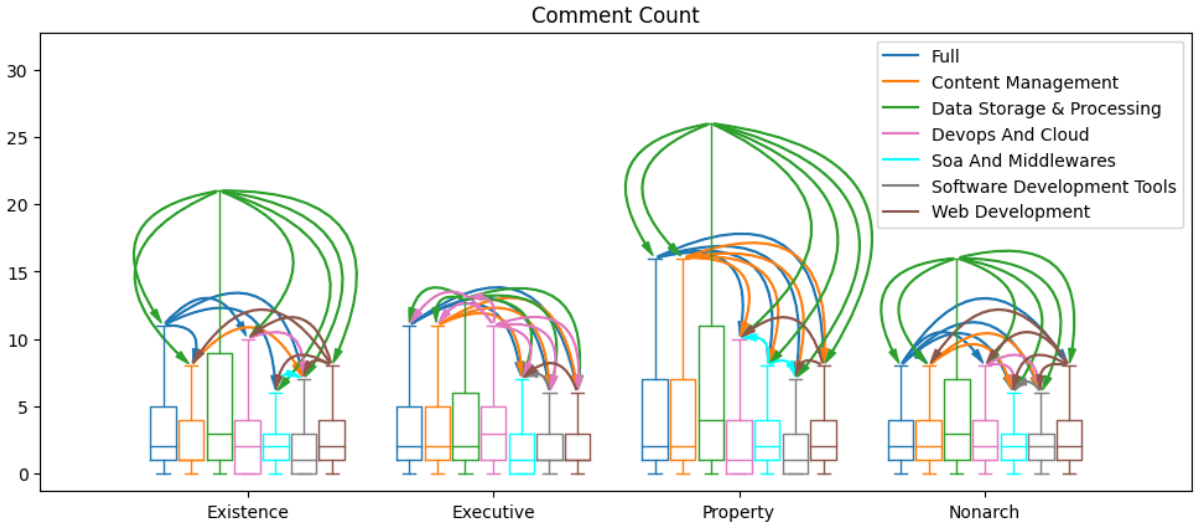


(a) Grouped by decision type first and subgrouped by domain.

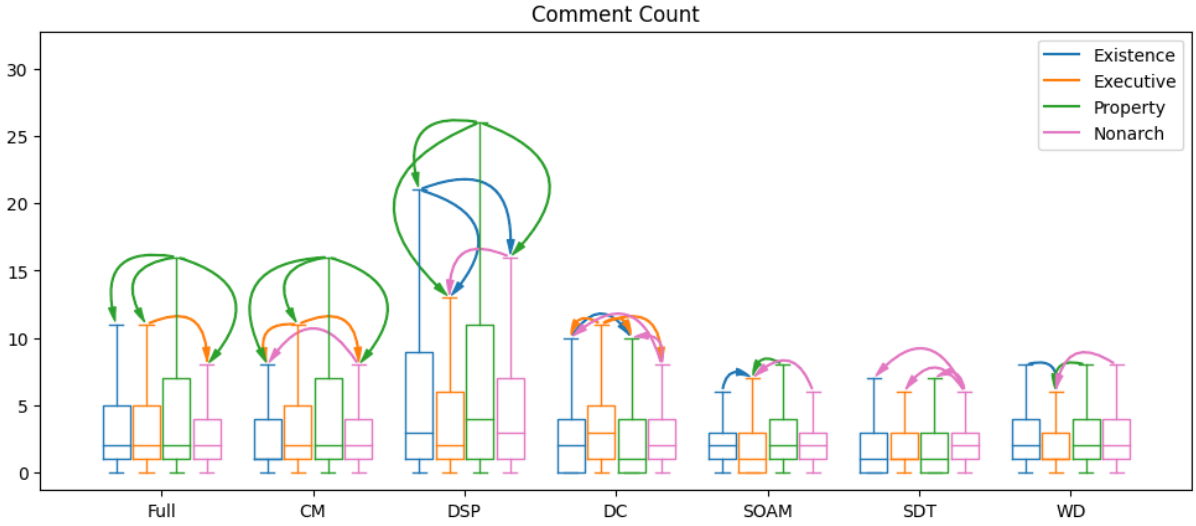


(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.4: Mann-Whitney test on the relations between description size, domain, and simple decision type contained in the issue.

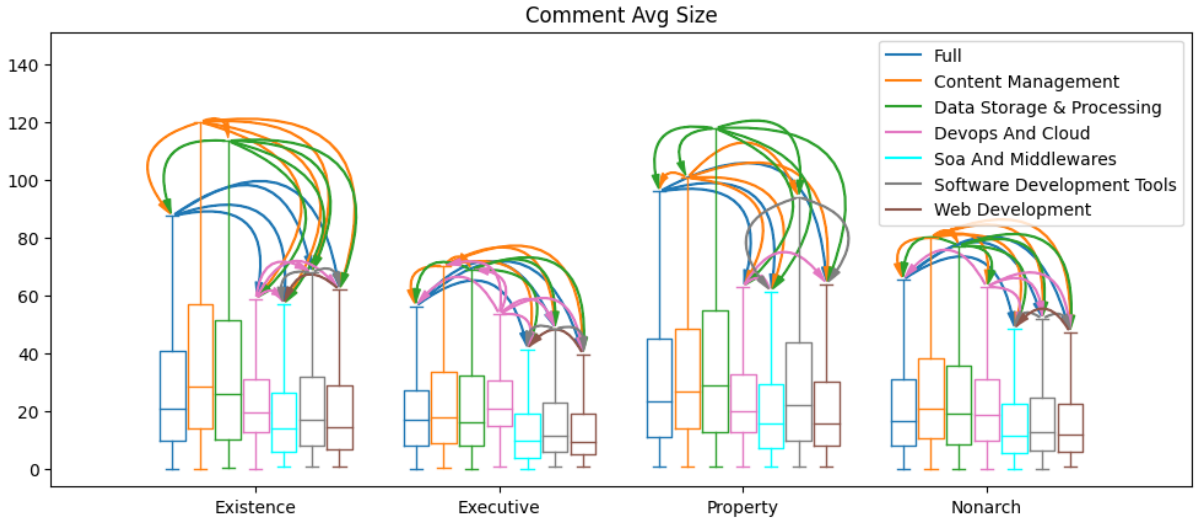


(a) Grouped by decision type first and subgrouped by domain.

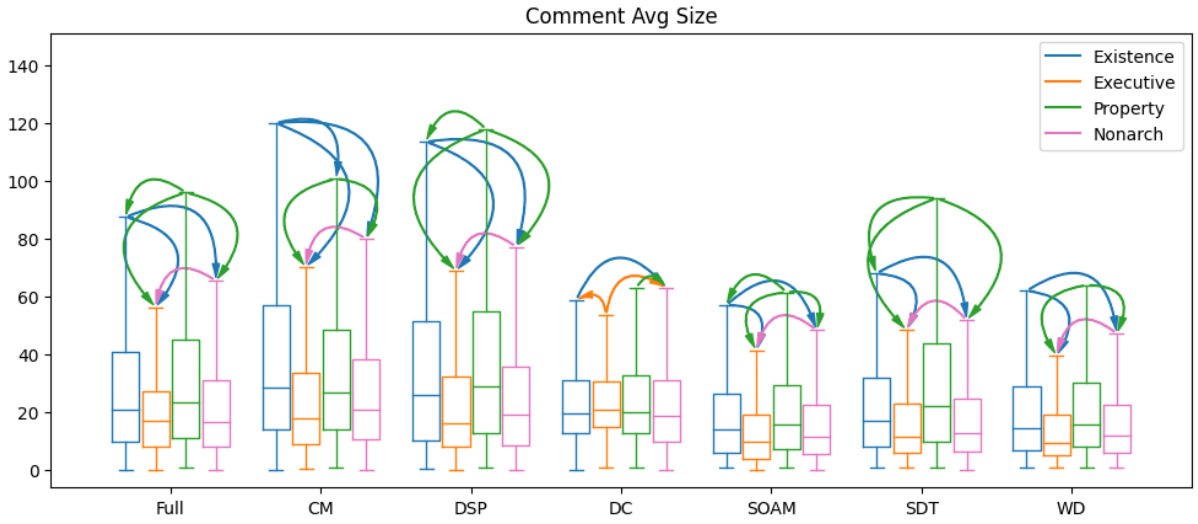


(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.5: Mann-Whitney test on the relations between comment count, domain, and simple decision type contained in the issue.



(a) Grouped by decision type first and subgrouped by domain.



(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.6: Mann-Whitney test on the relations between average comment size, domain, and simple decision type contained in the issue.

in DevOps and Cloud that this latter fact is not true. Also, in Content Management, **existence**'s comment size just barely manages to overtake **property**'s.

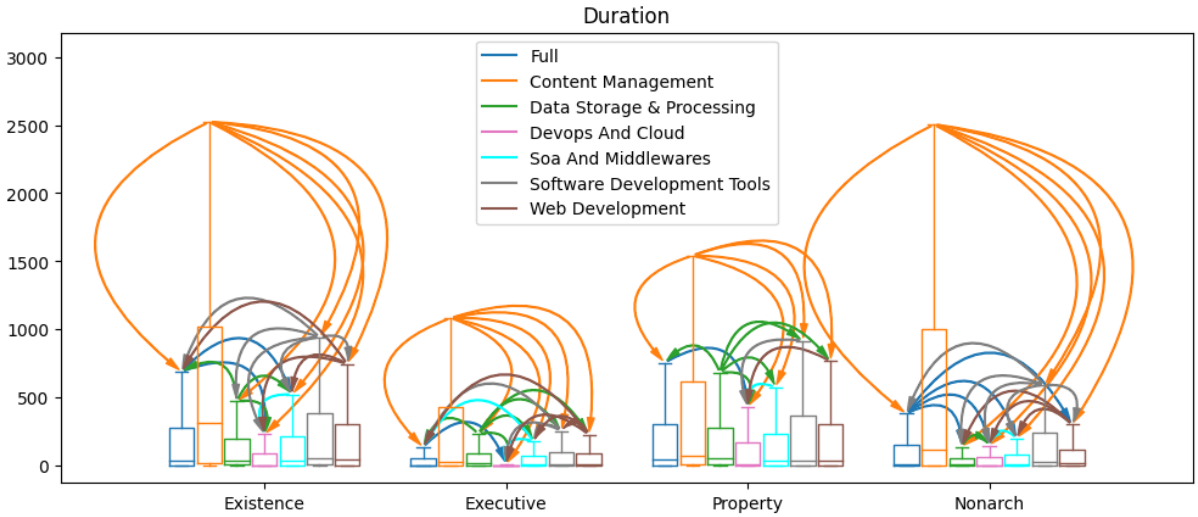
For the duration characteristic, in table 4.7, there is once again Content Management rising above the others, and this time with an interesting difference between it and the domain averages. Within Content Management, **existence-executive** issues take the longest, while in the domain average, they are near the lower end of all decision type combinations. Another interesting difference is that within Content Management, non-architectural issues are in the top half of issues that take the longest to resolve, while in the domain average, this is not the case. Both Content Management and the domain average seem to agree that any issue with **existence** in it takes longer than issues that do not.

Within the votes characteristic (figure 4.8, we will again use the Content Management domain data to draw conclusions, as other domains do not seem to use this feature as much as Content Management does. Again, issues that contain **existence** seem to get favoured for higher amounts of votes. Notably, non-architectural issues still get more votes than **executive** and **executive-property** issues do.

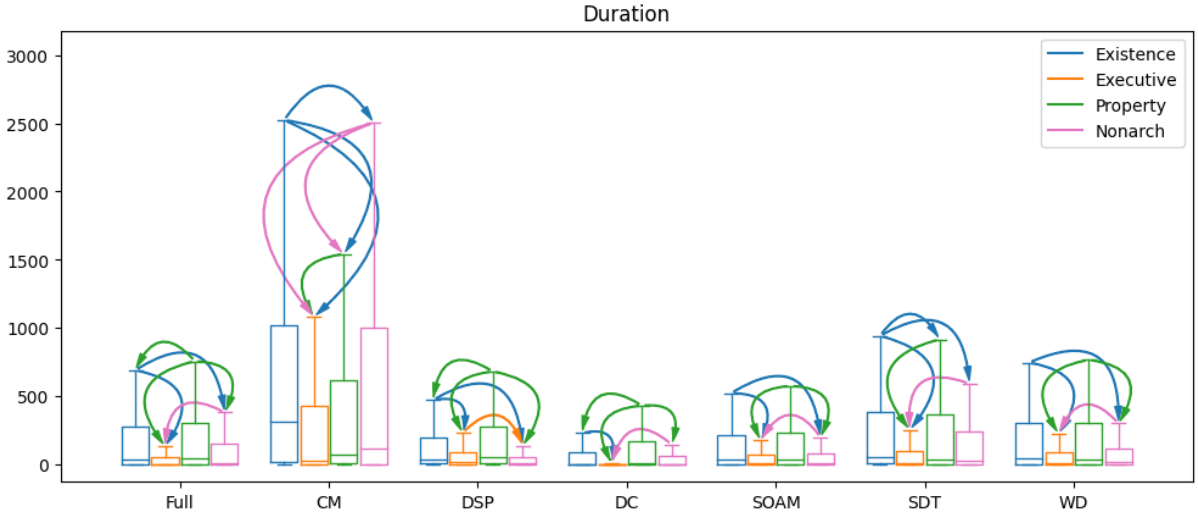
More domains seem to have participated in watches than just Content Management. A notable result here is that one again **existence** seems to get the most attention. Here again, though, the triple label issues are the highest in watches. **Executive** issues seem to get the least amount of watches, followed by non-architectural.

Key Takeaways for RQ3:
A pattern with the issues of type bugs emerges, along with the resolution values one might assume are associated with them. More interestingly, we have clear patterns of certain values for certain characteristics having implications for the architectural knowledge content of an issue. Such details that amplify an issue's chance to contain architectural knowledge are: being a parent issue (all); not being of type bug (all); being marked as won't fix or duplicate (**existence**, **property**, but not **executive**) or being marked done (**executive**); being marked as in-progress (all), open or reopened (only **existence** and **property**); having a great amount of comments (**property**); having large comments (**property** and **existence**); having a small description (**executive**); taking a very short amount of time (**executive**) or the opposite (**executive**, **property**).

Another interesting note to draw from these details is that **existence** and **property** seem to have more in common with each other than with **executive**, because they both pertain to the internal workings of the software being developed, while **executive** concerns itself with the external environment of the project.

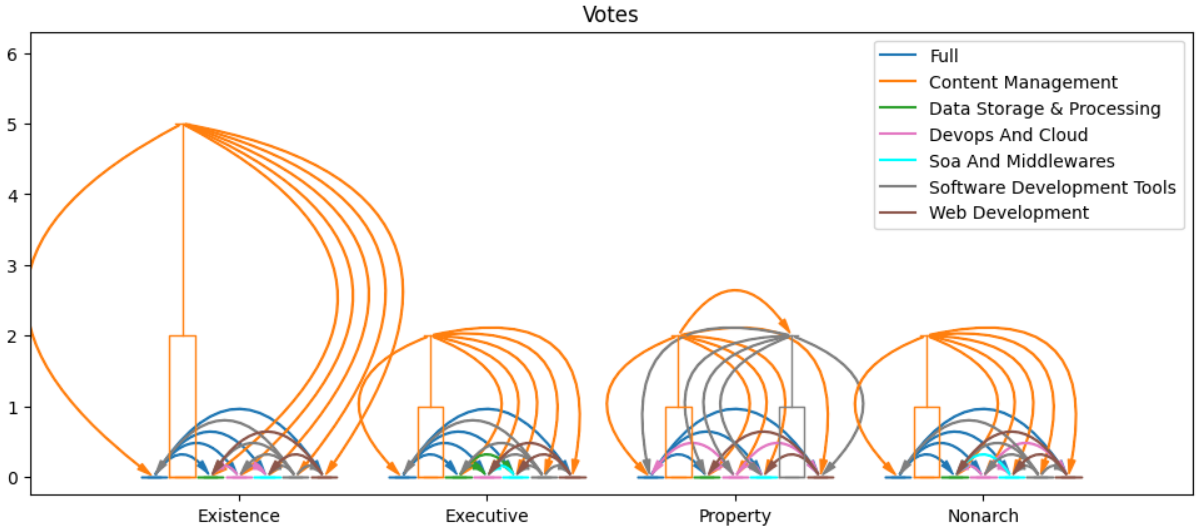


(a) Grouped by decision type first and subgrouped by domain.

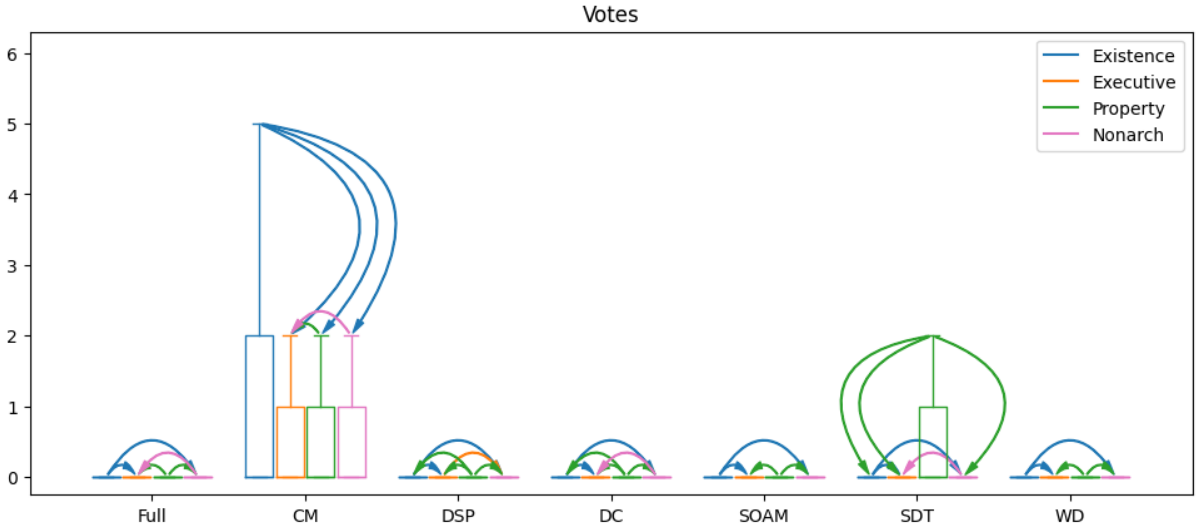


(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.7: Mann-Whitney test on the relations between duration, domain, and simple decision type contained in the issue.

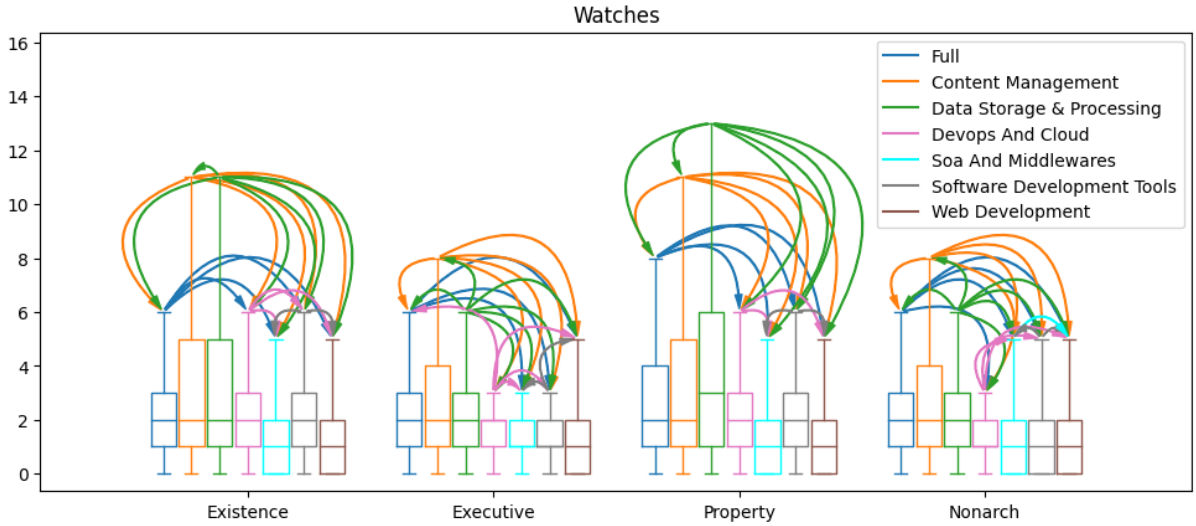


(a) Grouped by decision type first and subgrouped by domain.

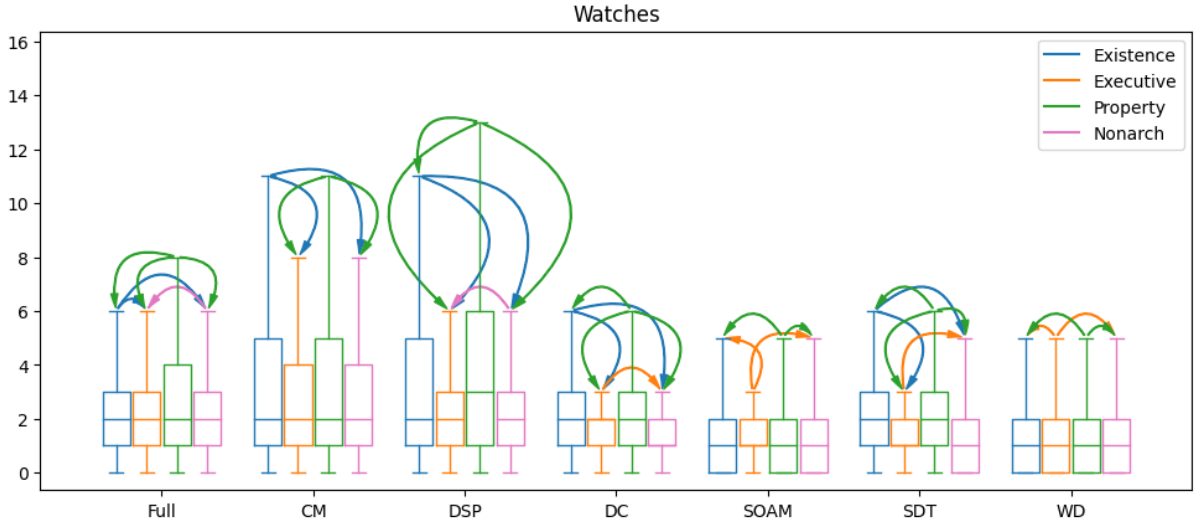


(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.8: Mann-Whitney test on the relations between votes, domain, and simple decision type contained in the issue.



(a) Grouped by decision type first and subgrouped by domain.



(b) Grouped by domain first and subgrouped by decision type. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

Figure 4.9: Mann-Whitney test on the relations between watches, domain, and simple decision type contained in the issue.

5 Discussion

In the results section, many initial connections were made between prevalence of certain issue labels under certain issue characteristic values. In this section, these connections will be further refined to conclusions, and from them, heuristics will be developed that can then be used by researchers and practitioners of software architecture.

It would seem that the software domain that an issue belongs to does have some influence over what decision types it is likely to contain. The resulting heuristic that we can formulate from this is was already written in the takeaways for RQ1: *in order to find both **executive** and **existence** decisions, one should investigate DevOps and Cloud projects. To supplement this with **property** decisions, investigate Data Storage and Processing.*

Next, issue characteristics within different domains do exhibit significant differences. Some of these differences likely relate to dominant project ecosystems within domains using certain Jira features more or less than others, or using different features (most notably: votes seemingly only being used by one domain and not the others). Other patterns can be combined with the results from RQ3 into more interesting conclusions.

In the key takeaways for RQ3, the heuristics for individual issue types have already been laid out. We can now connect them with the answers to the other research questions to formulate more detailed heuristics for each decision type, adding additional heuristic details from the domain-specific Mann-Whitney tests in RQ3 since we can narrow down each decision type to one domain:

- **For all types**: Examine parent issues that are not marked as bugs, within the domains DevOps and Cloud and Data Storage and Processing. Examine issues that are marked as still in progress. The more comments the issue has, the more likely it is architectural.
- **For Existence**: The all-types heuristic, with the change that the issue may also be a child issue instead of a parent but not an independent issue, with the additional filters that the issue's resolution is marked as won't fix or duplicate and the status is opened or reopened. The greater the description size, the smaller the comment count, the longer the duration, and the higher the watches, the more likely that it is **existence**. Search only within the domain DevOps and Cloud.
- **For Executive**: The all-types heuristic, with the additional filters that the issue is marked as done. The smaller the description, the greater the comment count and average size, the shorter the duration, the more likely the issue is **executive**. Search only within the domain DevOps and Cloud.
- **For Property**: The all-types heuristic, with the additional filters that the issue's resolution is won't fix or duplicate and the status is open or reopened. The greater the description size, comment count, comment average size and watches, the longer the duration, the more likely the issue is to be **property**. Search only within the domain Data Storage and Processing.

Other than these heuristics, there are several connections that jump out of the data. Existence receiving a lot of attention in the form of high watch counts seems to make sense, because existence decisions will be easy to grasp for more people than just the developers who know the system inside and out, in addition to being the decision type that will contain new features that people might be interested in. Additionally, looking at the intersected data in appendix C, it is not very surprising that the labels that contain more than one decision type get the most extreme values in many cases.

5.1 Implications for Researchers

5.1.1 Statistical Analysis of Related Projects

The statistical scripts that were written for this thesis can be found in the linked repository, and may be reused for other projects. Researchers can use this code, or use it as a base, to analyse both more issues and certain issues in more detail.

5.1.2 Expansion of the Dataset

Researchers can use the provided heuristics and tools to keep expanding the machine learning model training dataset, contributing to a richer potential for machine learning in this field of software architectural decisions in issue tracking systems.

5.1.3 More Detailed Statistical Analysis

As will be reiterated in the section on future work, the statistical code produced a lot of results, and much more detail can still be added, and more results can still be analysed in more detail, or in combination with other results. Researchers can

use this thesis as a base from which to start their own research into this field, and make use of the scripts, data and heuristics to advance knowledge in this field further.

5.2 Implications for Practitioners

5.2.1 Find Solutions for Problems

Using the UI tool, it is made easier to find solutions for problems a practitioner of software architecture may be facing. Using a combination of keywords and architectural types, the practitioner can search for relevant issues within the UI. The heuristics developed can also be used as a guideline to search for relevant issues outside of the UI tool, for example, within Jira itself, which has an advanced issue search filtering system, which even supports several issue characteristics as evaluated in this thesis, such as resolution, votes and watches.

5.2.2 Use Issue Tracker with More Precision

As professional software developers, it is highly likely that practitioners interested in this topic also use issue trackers such as Jira. Therefore, with the discoveries made in this thesis in mind, they might be able to better describe their issue if they are aware that it contains a certain decision type, for example in the following ways:

- If it is an **executive** decision, they know that the description does not need to be very long, and that the issue will likely not take much time.
- If it is a **property** decision, they know to invite and expect a larger-than-average discussion in the comments.
- If it is an architecturally relevant decision of any type, they will know to spend extra time considering if it can belong in any existing issue hierarchies, or requires a new one be made.

The heuristics, when kept in the back of the mind, can also be used to detect if an issue that a practitioner previously might not have thought was architectural, might turn out to contain design knowledge after all. For example, if there is more discussion than the original reporter expected, it might be a **property** decision, or if the issue racks up a substantial amount of votes, it might be an **existence** decision that many people are invested in. This will allow the practitioner to treat the issue with more appropriate diligence. This diligence may manifest as reclassifying the issue as a different type, or may be the motivation to formalise the knowledge in actual design documents.

6 Threats to Validity

6.1 External Validity

6.1.1 Status, Issue Type and Resolution

Notes about these three characteristics have been made earlier in the project, and as such, they deserve their own subsection in this list of threats to validity. It was unfeasible to analyse their full set of values for every domain, as there were multiple dozens for each characteristic. The strategy that was eventually used to select up to six values for each characteristic has been documented in the study design section, however, it is not infallible. Ideally, tests including these characteristics should be run on a level below that of the entire dataset: one single Jira instance, domain or even project, as it becomes less and less feasible to guarantee the consistency of the semantics behind each value, the bigger your analysed dataset is. The results of these characteristics are likely overly general: once you go down one or more levels and analyse these characteristics on a smaller scale, you may get different results, that will also be more applicable to your case. However, if the simplicity of the heuristic is also of import, the heuristics presented in this thesis are your best bet.

6.1.2 Heuristics as Deliverables

The status of the heuristics as solution to this difficulty to find architectural issues should not be taken as absolute. As is visible in the Mann-Whitney graphs and the wide variety of numbers in the chi-squared tables, there are broad ranges for almost every combination, and ultimately, logically, the architectural content of an issue does not depend directly on its characteristics. However, based on the comparative values of characteristics, using these heuristics should yield a greater amount of architectural issues compared to a random approach.

6.1.3 The Snowball Effect

A concern for using these heuristics to grow the machine learning model training dataset is that, if the statistics upon which these heuristics are based are themselves the result of a bias in the existing training set or the machine learning models themselves, then using these heuristics will likely only be exacerbated by this. Researchers who want to use these heuristics for this purpose should be aware of this and measure the benefits against the risks.

6.2 Construction Validity

6.2.1 Confidence Levels

Since the method of acquiring enough statistical data to perform analysis on was machine learning models, and access to the confidence levels of the model when it predicted labels for each issue is available, it was worth looking into potential differences between the regular dataset and a high-confidence dataset.

For this, a minimum confidence requirement was used for each decision type. These confidence values from the model come in the range of 0 to 1, so the minimum requirement as determined by the project's primary supervisor was 0.9 for **property** and **executive**, and 0.95 for **existence**. The results of this study are available in the same repository as linked above, namely `https://github.com/Shadania/design-decisions-stats`, with all related files having '_high_conf' in the filename. Most interesting to the reader is likely the pdf with all the same graphics as present in this paper, located at `https://github.com/Shadania/design-decisions-stats/blob/main/helper_scripts/highconf_tex/highconfonly.pdf`.

The result of this investigation was that there are no significant differences between the two versions that would have an impact on the results found in this paper. The amount of architectural issues discovered did go down significantly, but even there, the ratio of least common to most common decision type was preserved. In many chi-squared tables and Mann-Whitney box plots, small differences may be observed, but again, nothing so severe that it is likely to disturb the conclusions drawn in this thesis. However, any future research in this field using ML predictions of issue labels should likely go forward with only the high-confidence dataset.

6.2.2 Training Dataset

Dekker and Maarleveld (12) have already laid out threats to validity originating in the construction of the training dataset used for the original models: the disadvantage of letting a snowball effect speed up architectural issue discovery, that feeds directly back into the training dataset, is that one might be unknowingly reinforcing a bias of the model.

7 Conclusion and Future Work

In this thesis, first, a user interface to a complicated tool was improved, to help with creating a dataset to train machine learning models, so that statistical data could be generated. Next, a pipeline of statistical analysis scripts was set up so that this data could be processed in an orderly manner. Finally, these results were analysed in order to formulate heuristics to find patterns between architectural knowledge content and characteristics of issues in issue tracking systems. Several such heuristics were formulated, to be used in future research or development, to allow interested parties to find issues with a higher chance to be architectural.

7.1 Future Work

This thesis has laid a solid foundation for continued research into this topic, and provides researchers and practitioners with the tools to do so. However, improvements in several directions are possible, and in this section, they are laid out. The three identified categories are improvements to the web interface developed during the start of this thesis project, improvements to the statistical scripts themselves, and new routes to look for interesting statistics and patterns in.

7.1.1 ArchUI Improvements

- ArchUI has some basic statistical generation functionality. It could be improved in two ways:

    1. It could implement the statistics script that have been written and iteratively refined for this thesis, to offer more functionality.
    2. These scripts have been written in a largely modular way, so that the same function could be used with several different options. The UI could implement these options to give to the user to allow them to tailor their statistical research into their own database.
    3. The UI could improve on the user experience for this functionality, for example, through offering a progress bar, and display the reports in a more sleek, user-friendly way.

- The code of the UI has grown semi-organically as requirements were added and revised, and what it currently is was not particularly what was envisioned at the start of the several projects it has been a subject of. There have been attempts at refactoring, but the code still shows these organic origins.

- Now that heuristics with a statistical basis have been formulated, ArchUI should implement them. One of its strengths is already the snowball effect facilitation that its two areas of functionality create, and this would amplify that potential.

7.1.2 Statistical Improvements

The statistics code has been written in such a way that it should be easy to extend for new characteristics and modes, and should be able to accept new database with the same format without change. Though, here also, there is room for improvements.

- Not all potential issue characteristics were explored. Jira issues have many characteristics, direct or indirect, and more can always be investigated. It was difficult to generalise these characteristics for the complete dataset, so perhaps research in this direction could look into focusing on a specific domain, Jira instance or project, to identify characteristics relevant for the chosen area and focus thereon.

- There are currently many scripts, none of which require commandline arguments, but many of which require an in-text configuration to be modified to give different results. It may be difficult for an unfamiliar user to find the script they're looking for, despite the documentation available. Giving these scripts a graphical user interface through ArchUI would be one way to increase user-friendliness. Another, potentially quicker way would be to create a commandline interface.

7.1.3 New Statistical Avenues

This research has opened the doors to discovering and investigating more patterns within the data and draw more connections between issue characteristics and decision types. It has shown what might be traces of patterns that could be exploited to develop better heuristics. Below, some examples of this potential that should be looked into:

- There is a similar pattern in the result data for bugs and architectural content, and some bug-related resolutions and architectural content. Per issue type, the interaction of the resolution characteristic and decision type should be explored.

- Many parent issues are architectural, and slightly more children than average are architectural also. A new avenue to explore is to discover if there is a link between a parent

issue of a certain decision type and its children: for example, does a **property**-related parent issue imply **property**-related children?

- Jira allows issues to be linked together in more ways than a parent issue to a child. These relationships should also be investigated, for similar patterns as described in the item above.
- Due to the high-level nature of this project, mostly only high-level results were evaluated. There is potential in doing a similar study but on the level of a single Jira instance, or even project. This would solve issues such as the one encountered with the status, resolution and issue_type characteristics, where values might have been used in different ways across different projects. An interesting first step in this direction would be to evaluate whether there are any significant differences between projects within one domain, and to analyse the outliers closer.
- Not all the results generated by the scripts were able to be analysed and used due to time constraints. Please see the repository for an overview of all generated results. Potentially useful data may be hiding in there.

7.1.4 Evaluations

This thesis has brought results and innovations, but due to time constraints, not many results have been thoroughly tested. Therefore, the author would like to propose the following ways to validate the results further:

- The ArchUI interface should be evaluated for its ease of use amongst practitioners and researchers not within the stakeholder group evaluated for requirements and feature requests. From this extended group, more requirements can be gathered, so that ArchUI may become a more generally available tool for greater groups.
- The developed heuristics should be tested, both statistically and practically. This would benefit from them having been implemented within the UI first.

References

[1] Zoya Alexeeva, Diego Perez-Palacin, and Raffaela Mirandola. Design decision documentation: A literature overview. In *Software Architecture*, pages 84–101. Springer International Publishing, 2016.

[2] Shikhar Bharadwaj and Tushar Kadam. GitHub issue classification using BERT-style models. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*. ACM, may 2022.

[3] Georg Buchgeher, Rainer Weinreich, and Heinz Huber. A platform for the automated provisioning of architecture information for large-scale service-oriented software systems. In *Software Architecture*, pages 203–218. Springer International Publishing, 2018.

[4] Rafael Capilla and Francisco Nava. Extending software architecting processes with decision-making activities. In *Balancing Agility and Formalism in Software Engineering*, pages 182–195. Springer Berlin Heidelberg, 2008.

[5] Arjan Dekker and Jesse Maarleveld. Mining for architectural design decisions in issue tracking systems using deep learning approaches, July 2022.

[6] Wilhelm Hasselbring. Software architecture: Past, present, future. In *The Essence of Software Engineering*, pages 169–184. Springer International Publishing, 2018.

[7] Sebastian Herold, Andreas Metzger, Andreas Rausch, and Heiko Stallbaum. Towards bridging the gap between goal-oriented requirements engineering and compositional architecture development. In *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*. IEEE, may 2007.

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

[9] Alyssa Josephs, Fabian Gilson, and Matthias Galster. Towards automatic classification of design decisions from developer conversations. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, mar 2022.

[10] Jan Keim, Sophie Schulz, Dominik Fuchß, Claudius Kocher, Janek Speit, and Anne Koziolek. Trace link recovery for software architecture documentation. In *Software Architecture*, pages 101–116. Springer International Publishing, 2021.

[11] Philippe Kruchten, Patricia Lago, and Hans van Vliet. Building up and reasoning about architectural knowledge. In *Quality of Software Architectures*, pages 43–58. Springer Berlin Heidelberg, 2006.

[12] Jesse Maarleveld and Arjan Dekker. Developing deep learning approaches to find and classify architectural design decisions in issue tracking systems. Master's thesis, Rijksuniversiteit Groningen, https://fse.studenttheses.ub.rug.nl/31368/, 2023.

[13] Jesse Maarleveld, Arjan Dekker, Sarah Druyts, and Mohamed Soliman. Maestro: A tool to find and explore architectural design decisions in issue tracking systems. 2023. To be published.

[14] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, New York, 1995.

[15] M. Luciana Roldán, Silvio Gonnet, and Horacio Leone. A model for capturing and tracing architectural designs. In *Advanced Software Engineering: Expanding the Frontiers of Software Technology*, pages 16–31. Springer US, 2006.

[16] Yves Schneider, Axel Busch, and Anne Koziolek. Using informal knowledge for improving software quality trade-off decisions. In *Software Architecture*, pages 265–283. Springer International Publishing, 2018.

[17] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. Recovering architectural design decisions. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, apr 2018.

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

[19] Minh Tu Ton That, Salah Sadou, Flavio Oquendo, and Régis Fleurquin. Preserving architectural decisions through architectural patterns. *Automated Software Engineering*, 23(3):427–467, oct 2014.

A Maestro Tool Paper

Maestro: A Tool to Find and Explore Architectural Design Decisions in Issue Tracking Systems

Jesse Maarleveld, Arjan Dekker, Sarah Druyts, and Mohamed Soliman

University of Groningen (RUG), Groningen, The Netherlands
{j.maarleveld,a.j.dekker.5,s.druyts}@student.rug.nl, m.a.m.soliman@rug.nl

Abstract. Software engineers commonly re-use architectural design decisions (ADDs) from their previous experience. However, in practice, software engineers still depend on adhoc mechanisms to re-use ADDs. Recent studies show that software engineers discuss ADDs in issue tracking system, which could be useful for software engineers to make new ADDs. Nevertheless, it is rather challenging to find ADDs among the big amount of issues in issue trackers. Therefore, we introduce Maestro, an open source tool for finding, annotating, and exploring ADDs in issue tracking systems. The tool allows researchers and practitioners to find and analyze issues containing ADDs in issue trackers. Maestro provides annotation mechanisms, deep learning components, keywords-based search engine and a user-interface that can be easily used by researchers and practitioners to find and analyze ADDs in issue trackers.

Keywords: Architectural design decisions · issue tracking system.

1 Introduction

Software engineers tend to reuse the knowledge from previously made Architectural Design Decisions (ADDs) [14], such as ADDs on components design (e.g. through patterns ([5])), technology ADDs [20], and ADDs on tactics to address quality requirements (e.g. authentication mechanisms as security tactics) [2]. For instance, software engineers can learn from the drawbacks (e.g. performance issues) of solutions decided in previous ADDs. The re-use of knowledge from previous ADDs could help software engineers to effectively design new systems and mitigate risks.

While re-using ADDs could be useful in practice, empirical studies show that software engineers do not commonly document ADDs [14]. For instance, researchers proposed a wide variety of tools to manage and document ADDs [6, 23, 24]. However, software engineers still tend to maintain their knowledge on ADDs in their head (i.e. tacit) without explicit documentation [6]. On the other hand, software engineers communicate and discuss ADDs *informally* to resolve issues (e.g. new features[1] or improvements[2]) in issue tracking systems

[1] https://issues.apache.org/jira/browse/HADOOP-13944
[2] https://issues.apache.org/jira/browse/CASSANDRA-12245

(e.g. Jira) [19, 3]. We call issues containing such discussions *architectural issues*. The discussions on ADDs in architectural issues contain useful knowledge, which software engineers could potentially re-use to make new ADDs.

While architectural issues could potentially be useful for software engineers, they are not tagged by software engineers [19], which make them hard to find and explore in between the vast majority of issues on programming and bugs. Therefore, researchers utilised different approaches (e.g. machine learning [3], source code analysis [19], and qualitative analysis [19]) to find and explore architectural issues, each with different pros and cons. However, the diversity of the different approaches require researchers and practitioners to execute each approach separately, and possibly manually combine their results to effectively find and explore architectural issues. To execute each approach separately is a complex, error prone and time-consuming process, which require expertise in different fields like machine learning and qualitative analysis.

In this paper, we propose Maestro: An open source tool[3] to find and explore ADDs in issue tracking systems. Maestro combines four different approaches to find and explore ADDs in a single process: keyword-based searches, deep learning, qualitative analysis, and statistical analysis. In addition, Maestro allows importing results from other approaches such as source code analysis. In Maestro, we distinguish between different types of ADDs according to Kruchten et al. [13]: existence (component related), executive (process and technology related), and property (quality related). Maestro is designed to be extensible and easy to use for both researchers and practitioners. For instance, software engineering researchers can train and run deep learning models without expertise on programming deep learning models. Maestro can be deployed remotely or locally, which provides flexibility for researchers and practitioners to run the tool.

The rest of the paper is organised as follows: In Section 2, we discuss the use cases of Maestro. In Section 3, we discuss the architecture of Maestro. We explain our experiences and evaluation of Maestro in Section 4, and compare it with related work in Section 5. Finally, we conclude the paper in Section 6.

2 Use Cases

Maestro serves both researchers and practitioners to find and explore ADDs in issue tracking systems. Researchers can use Maestro for empirical analysis; practitioners can use Maestro to re-discover and re-use architecetural knowledge. Fig 1 shows an overview of the use cases supported by Maestro and their relationships. We explain each use-case below:

UC1 Select candidate issues for qualitative analysis: Researchers can select certain issues to be manually analysed (in UC2). The nomination of the selected issues can come from different sources: 1) predictions made by deep learning classifiers (in UC4). 2) issues resulting from keywords-searching (in

[3] Available from: `https://github.com/mining-design-decisions/Maestro`

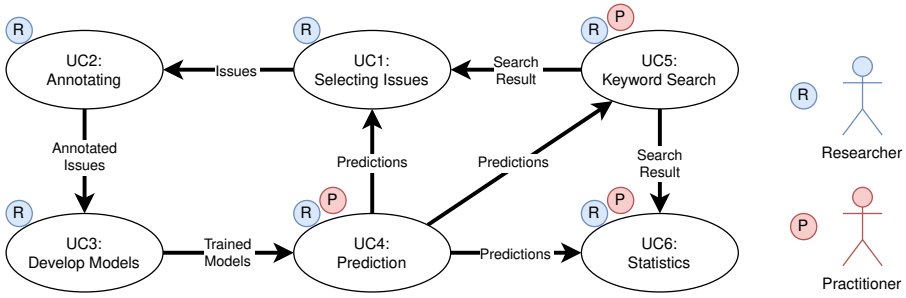


Fig. 1. Use cases supported by Maestro, annotated with relevant actors per use case. Arrows show how results from one use case (or activity) are used by other use cases.

UC5). 3) issues identified from other tools (e.g. source code analysis [19, 18]), and 4) issues selected randomly similar to Bhat et al. [3].

UC2 **Annotate issues with types of ADDs**: Researchers can analyse selected issues (from UC1) using qualitative methods (e.g. grounded theory [21]), and annotate them based on the types of ADDs within issues. Using the tool, multiple remotely located researchers can discuss types of ADDs using an online conversation associated with each issue. The UI provides the researchers with the summaries and descriptions of issues, the assigned types of ADDs, and a discussion thread per issue. The conversations between researchers can be used incrementally to create a coding book for annotating architectural issues. Furthermore, the tool supports researchers to calculate agreement measures such as Kappa [9] to ensure high quality of the qualitative analysis. The annotated issues can be directly used to develop new deep learning models (in UC3).

UC3 **Develop deep learning models to identify types of ADDs in issues**: First, researchers can design classifiers by choosing from different types of feature generation (e.g. Word embedding [15] and Word Frequency), deep learning architectures (e.g. RNN [12, 8, 7], CNN [17], and BERT [11]), which can be automatically tuned using the flexible user interface of the tool. Second, researchers can train designed classifiers using the annotated issues (from UC2), and compute their accuracy (e.g. in terms of F_1 score) to automatically identify types of ADDs in issues.

UC4 **Predict types of ADDs in issues**: Both practitioners or researchers can use the trained classifiers (from UC3) to predict types of ADDs in new, previously un-annotated, issues. Specifically, practitioners can find past ADDs in issues of existing projects, understand their rationale, and re-use their knowledge to make new ADDs. Researchers could further analyse these issues using qualitative analysis (in UC2) or statistical analysis (in UC6).

UC5 **Search for ADDs using keywords**: Both practitioners or researchers can search for architectural issues using classical keywords-based search (i.e. information retrieval). Moreover, the tool facilitates filtering search results based on the predictions of classifiers (from UC4). In this way, practitioners

could effectively find issues that discuss certain types of ADDs. At the same time, researchers can focus their qualitative and statistical analysis (in UC2 and UC6) on issues that discuss certain types of ADDs.

UC6 Perform statistical analysis on ADDs: Researchers and practitioners could perform statistical analysis on architectural issues. For example, practitioners could determine the duration of issues that involve certain types of ADDs. This can help practitioners to estimate the duration of future ADDs based on their type. As another example, researchers might be interested to determine the amount of knowledge on certain types of ADDs in the descriptions and comments of architectural issues.

3 Architecture of Maestro

Maestro consists of four layers, each contains multiple components. The logical architecture is depicted in Figure 2, and the physical architecture in Figure 3. We explain below each layer in more details:

- The **Persistence Layer** contains four different databases: 1) a database that contains data on issues (e.g. summary and description), which we based on the dataset from Montgomery et al. [16]. 2) a database that contains data related to the manual annotation of issues (e.g. manual labels and discussions between researchers), and all deep learning related data (e.g. trained models, their configurations, performance scores), 3) a database that contains cached statistics data, and 4) a database for usernames and passwords.
- The **Data Access Layer** provides secure access to the databases using authentication tactics. Furthermore, it contains components that can update the issues database with new issues from issue trackers (current only Jira is supported) to support the extensibility of the system. We re-used the component created by Montgomery et al. [16], and enhanced it to be extensible.
- The **Processing Layer** contains two major components: 1) The *Keywords Search Engine* provides a centralised API for performing keyword searches (UC5) using Apache Lucene, which allows re-use of pre-computed indices. 2) The *Deep Learning Manager* acts as the backend for all deep learning related functionality outlined in UC2 and UC3. The deep learning was designed to be extensible. In Fig 2, every pipeline makes use of one or more entities. New entities, such as new feature generators or neural networks, can be easily added by adding new entity classes which are instantiated through factories.
- The **User Interface** provides an interface for the user to fulfil all use-cases in Section 2. For instance, to achieve UC3, the UI presents different options for each deep learning model and provides a user-friendly interface to provide parameter values. Through the UI, researchers could initiate the training of machine learning models, and view accuracy scores in a concise overview. Moreover, researchers could manually view and classify issues (UC2). Further details on the UI can be viewed in our video[4]. The UI is designed according to

[4] `https://www.youtube.com/watch?v=sztY5it5Lb4`

the Model-View-Controller (MVC) pattern, and depends on the processing layer and the data access layer (see Figure 2).

Components can be deployed locally or remotely (Fig 3), allowing data centralisation and offloading of computationally intensive tasks to other devices.

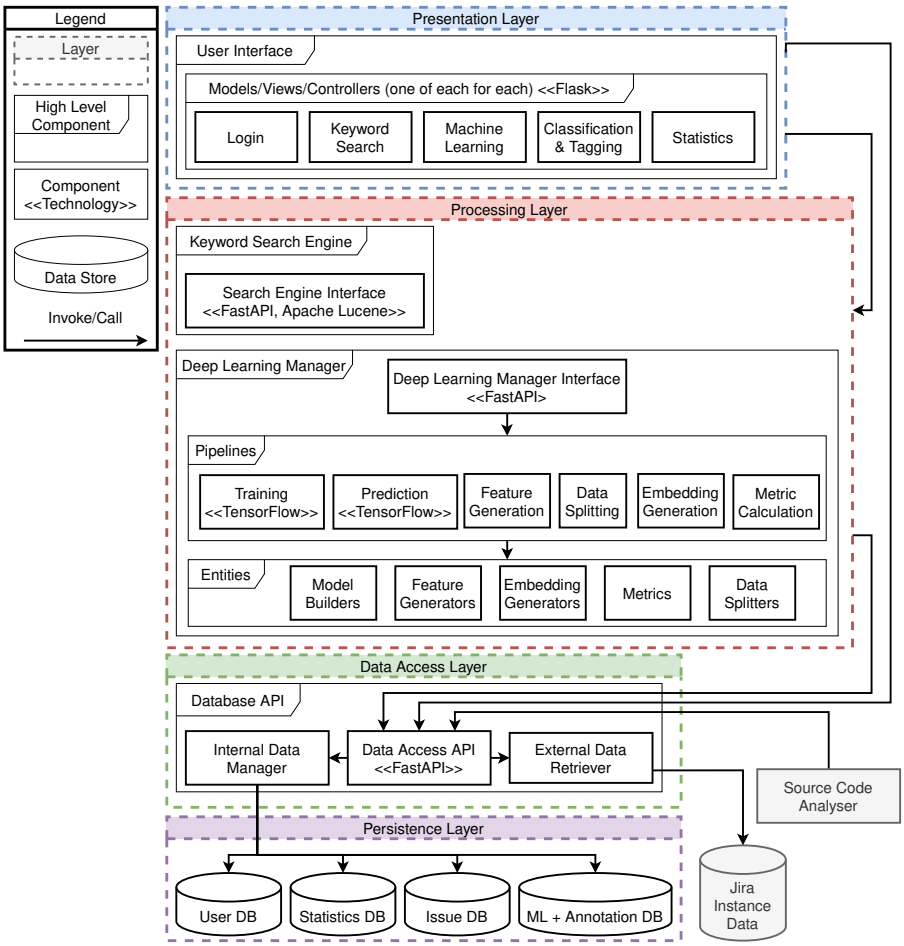


Fig. 2. The logical architecture of Maestro. The "high level components" are larger components with smaller sub-components.

4 Research Process to Develop Maestro

Maestro is a result of a research project spanning more than 2 years of efforts [10] that aims to explore ADDs in issue tracking systems. The four authors of

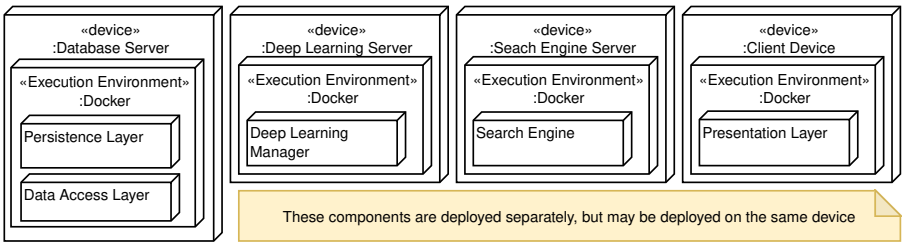


Fig. 3. The physical architecture of Maestro.

this paper, as well as two other independent researchers, participated in this project. Our research follows an action research method [1], where researchers investigated the problem of finding and exploring architectural issues in issue trackers, and simultaneously developed approaches to find and analyse architectural issues. In detail, we followed four phases, each consists of an action and an evaluation steps. We explain below each phase and step, and associate them to the use-cases (UC) in Section 2. We explain how these phases lead to the development of Maestro, and illustrate how it can be used in research.

- **Phase 1 - Random sampling to find architectural issues**:
  Action: We selected a random sample of 400 issues from six different open-source projects, and analysed them using qualitative analysis [21].
  Evaluation: The percentage of architectural issues range between 10-15% of the random sample, which shows that random sampling is not an effective approach to find ADDs in issue trackers.
- **Phase 2 - Keywords-search and source code analysis**:
  Action: Because random sampling was ineffective to find architectural issues, we experimented with two further approaches: searching using keywords from literature (**UC5**), and source code analysis [19]. Using both approaches, we selected 2179 candidate issues (**UC1**) from six open source projects from the Big Data domain (e.g. Apache Hadoop) to be manually analysed using qualitative analysis. For each issue, we downloaded its title and description in an excel sheet, and annotated the types of ADDs in their descriptions according to Kruchten et al. [13]: Existence, property and executive. Disagreements between researchers were discussed in separate meetings.
  Evaluation: Keywords searching and source code analysis were effective to find existence ADDs (precision $> 50\%$), but suffered from low precision to find property and executive ADDs. Moreover, during the qualitative analysis, we realised that it is challenging to annotate large number of issues using Excel sheets, because some issues are long and contain formatting symbols, which cannot be correctly visualised. It was also challenging to track our discussions on issues during our meetings. These discussions were important to write and improve our coding book to annotate ADDs in issue trackers.
- **Phase 3 - Machine learning to find architectural issues**:
  Action: Because keywords-search and source code analysis were not effec-

tive to find property and executive architectural issues, we trained different deep learning models to automatically classify architectural issues (**UC3**). We then used the model with the best accuracy (i.e. "BERT" model) to predict the types of issues (**UC4**), which have not been previously manually analysed. Accordingly, We sorted the issues identified from "BERT" model depending on the confidences obtained from the model to analyse manually (**UC1**). We developed the user interface of the tool to display and sort list of issues based on the confidences generated by deep learning models. Furthermore, we developed a dedicated user interface to annotate and tag issues based on the types of ADDs in their description (**UC2**).
Evaluation: The tool showed significant usefulness to annotate issues, because researchers (allocated remotely) could directly view, discuss and classify issues in one process. According to our experience, using the tool was better than relying on excel sheets, especially in visualising long and complex issues. Moreover, the tool allows to discuss issues, and instantly add issues to the training set without any need to run other scripts or upload data, which prevent faults such as forgetting to include issues or inserting duplicate issues (i.e. the tool provides a consistent overview of all labelled issues for all users). Additionally, during annotations, the tool allows adding tags to issues, which helped us to mark issues that require a second opinion on their classification, and enabled us to track information about who annotated which issues, and how these issues were found (e.g. using keywords searching – **UC5**). This tagging functionality helped us to more easily identify groups of potentially miss-annotated issues. Furthermore, the UI brings notable enhancements to train deep learning models. Previously, we had to manually create configurations for each model, which was error-prone and tedious. However, the UI now clearly presents all available options for each model to facilitate creation, training and evaluation. Using this new functionality of the tool, we performed UC2-UC4 in 3 iterations to expand our dataset to reach 2210 architectural issues and 2903 non-architectural issues.

– **Phase 4 - Find architectural issues from different domains**:
Action: In the previous phases, we explored ADDs in six open-source projects from the Big Data domain. In this phase, we explore ADDs in projects from different domains other than Big Data. Thus, we re-used a recent dataset from Montgomery et al. [16], which contains more than 2.7 million Jira issues from 1352 projects that belong to six different domains including Big Data, Cloud Computing, SOA, and DevOps. We trained and executed the best performing model (i.e. "BERT") to identify architectural issues and predict the types of ADDs in all issues in the dataset (**UC4**). We also developed a statistical analysis functionality in the tool (**UC6**) to visualise the types of ADDs in the different domains, as well as the characteristics of architectural issues such as time to resolve and the amount of discussion in comments.
Evaluation: Using the tool, we identified 250,708 architectural issues from the six domains. Moreover, we determined the most common types of ADDs per domain, and compared characteristics of architectural issues per domain. For example, issues that discuss property ADDs were most involved and took

longer time to resolve. The statistical functionality in the tool (**UC6**) shows its usefulness to explore ADDs in a massive number of architectural issues.

5 Related Work

Several traditional architectural knowledge management tools have been previously proposed [22]. These tools store and document ADDs in repositories and templates, which need to be manually populated. On the other hand, our proposed tool Maestro focus on ADDs discussed in issue tracking systems.

The closest tool to Maestro is ADeX [4], which can classify architectural issues using machine learning. Moreover, ADeX can recommend developers for making certain ADDs based on personal expertise. While both tools ADeX and Maestro aim to find and explore ADDs in issue trackers, our proposed tool Maestro is different than ADeX in the following points:

- Maestro allows researchers to apply qualitative analysis (in UC2), and add manually classified issues to the training dataset. Moreover, Maestro supports keywords-based searches (in UC5), which allows researchers to easily expand their dataset of architectural issues through a snowballing process. This process is not supported by ADeX.
- Maestro provides a user-friendly UI to train and evaluate *new deep learning models* (in UC3 and UC4), which can help researchers to evolve models for classifying architectural issues. This flexibility is not provided by ADeX, which provides pre-trained machine learning models for classification. The accuracy of the pre-trained model is fixed based on Bhat et al. [3].
- Maestro has been evaluated on a large dataset of issues with 2.7 million issues from different domains, which show its scalability and usefulness to run on projects from different domains. In contrast, ADeX has been applied on two open source projects.
- Maestro is open source[5] and is designed to be extended by other researchers or practitioners. In contrast, the source code of ADeX is not referenced by the authors of ADeX.

6 Conclusion

We developed Maestro, an open source tool for finding, and exploring architectural issues that discuss design decisions. Our experience with Maestro showed its usefulness to find and annotate 5113 issues, and develop deep learning models that automatically classified 250,708 architectural issues. Contrary to existing tools, Maestro supports researchers to find and annotate architectural issues through keywords searching, deep learning models and snowballing. Our future work focuses on evaluating Maestro with practitioners to evaluate its usefulness to re-use ADDs from issue trackers. Furthermore, we aim to use Maestro to further expand our dataset with new issues from different projects, and different issue trackers. This can improve the accuracy and generalizability of Maestro.

[5] `https://github.com/mining-design-decisions/Maestro`

References

1. Baskerville, R.L., Wood-Harper, A.T.: A Critical Perspective on Action Research as a Method for Information Systems Research. In: Willcocks, L.P., Sauer, C., Lacity, M.C. (eds.) Enacting Research Methods in Information Systems: Volume 2, pp. 169–190. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-29269-4_7, https://doi.org/10.1007/978-3-319-29269-4_7
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional (2003), google-Books-ID: mdiIu8Kk1WMC
3. Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., Matthes, F.: Automatic extraction of design decisions from issue management systems: A machine learning based approach. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **10475 LNCS**, 138–154 (2017). https://doi.org/10.1007/978-3-319-65831-5_10, publisher: Springer Verlag ISBN: 9783319658308
4. Bhat, M., Tinnes, C., Shumaiev, K., Biesdorf, A., Hohenstein, U., Matthes, F.: ADeX: A Tool for Automatic Curation of Design Decision Knowledge for Architectural Decision Recommendations. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). pp. 158–161 (Mar 2019). https://doi.org/10.1109/ICSA-C.2019.00035
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley, Chichester, UK (1996)
6. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A.: 10 years of software architecture knowledge management: Practice and future. Journal of Systems and Software **116**, 191–205 (Sep 2015). https://doi.org/10.1016/j.jss.2015.08.054
7. Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the Properties of Neural Machine Translation: Encoder-Decoder Approaches (Oct 2014). https://doi.org/10.48550/arXiv.1409.1259, arXiv:1409.1259 [cs, stat]
8. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling (Dec 2014). https://doi.org/10.48550/arXiv.1412.3555, arXiv:1412.3555 [cs]
9. Cohen, J.: A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement **20**(1), 37–46 (Apr 1960). https://doi.org/10.1177/001316446002000104, publisher: SAGE Publications Inc
10. Dekker, A., Maarleveld, J.: Mining for Architectural Design Decisions in Issue Tracking Systems using Deep Learning Approaches. MSc Internship Report, University of Groningen, Groningen (2022), https://fse.studenttheses.ub.rug.nl/28689/
11. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (May 2019). https://doi.org/10.48550/arXiv.1810.04805, http://arxiv.org/abs/1810.04805, arXiv:1810.04805 [cs]
12. Hochreiter, S., Schmidhuber, J.: Long Short-term Memory. Neural computation **9**, 1735–80 (Dec 1997). https://doi.org/10.1162/neco.1997.9.8.1735
13. Kruchten, P.: An ontology of architectural design decisions in software intensive systems. 2nd Groningen workshop on software variability (2004)
14. Manteuffel, C., Avgeriou, P., Hamberg, R.: An exploratory case study on reusing architecture decisions in software-intensive system projects. Journal of Systems and Software **144**, 60–83 (Oct 2018). https://doi.org/10.1016/j.jss.2018.05.064

15. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space (Sep 2013). https://doi.org/10.48550/arXiv.1301.3781, http://arxiv.org/abs/1301.3781, arXiv:1301.3781 [cs]
16. Montgomery, L., Lüders, C., Maalej, W.: An Alternative Issue Tracking Dataset of Public Jira Repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories. pp. 73–77 (May 2022). https://doi.org/10.1145/3524842.3528486, arXiv:2201.08368 [cs]
17. Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., Grundy, J.: Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability. ACM Transactions on Software Engineering and Methodology **28**(3), 15:1–15:45 (Jul 2019). https://doi.org/10.1145/3324916
18. Shahbazian, A., Kyu Lee, Y., Le, D., Brun, Y., Medvidovic, N.: Recovering Architectural Design Decisions. Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018 pp. 95–104 (Jul 2018). https://doi.org/10.1109/ICSA.2018.00019, publisher: Institute of Electrical and Electronics Engineers Inc. ISBN: 9781538663981
19. Soliman, M., Galster, M., Avgeriou, P.: An Exploratory Study on Architectural Knowledge in Issue Tracking Systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **12857 LNCS**, 117–133 (2021). https://doi.org/10.1007/978-3-030-86044-8_8/FIGURES/3, arXiv: 2106.11140 Publisher: Springer Science and Business Media Deutschland GmbH ISBN: 9783030860431
20. Soliman, M., Riebisch, M., Zdun, U.: Enriching Architecture Knowledge with Technology Design Decisions. In: 2015 12th Working IEEE/IFIP Conference on Software Architecture. pp. 135–144 (May 2015). https://doi.org/10.1109/WICSA.2015.14
21. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering. pp. 120–131. ICSE '16, Association for Computing Machinery, New York, NY, USA (May 2016). https://doi.org/10.1145/2884781.2884833
22. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Ali Babar, M.: A comparative study of architecture knowledge management tools. Journal of Systems and Software **83**(3), 352–370 (Mar 2010). https://doi.org/10.1016/j.jss.2009.08.032
23. Weinreich, R., Groher, I.: Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review. Information and Software Technology **80**, 265–286 (Dec 2016). https://doi.org/10.1016/j.infsof.2016.09.007
24. Weinreich, R., Groher, I., Miesbauer, C.: An expert survey on kinds, influence factors and documentation of design decisions in practice. Future Generation Computer Systems **47**, 145–160 (Jun 2015). https://doi.org/10.1016/j.future.2014.12.002

B Projects Per Software Domain

Web Development	Devops And Cloud	Soa And Middlewares	Data Storage & Processing	Content Management	Software Development Tools
Apache-TOMEE	RedHat-ACM	Apache-SM	Apache-CASSANDRA	Jira-JRACLOUD	RedHat-CRW
Apache-DELTASPIKE	RedHat-TRACING	RedHat-JBWS	Spring-BATCH	Apache-SOLR	Apache-MNG
RedHat-UNDERTOW	Apache-AURORA	Apache-SMX4	Apache-AMBARI	Apache-CONNECTORS	Spring-ROO
RedHat-ERRAI	RedHat-OADP	Apache-FELIX	Apache-BEAM	Jira-JSDCLOUD	Jira-SRCTREE
Apache-WICKET	Apache-STRATOS	Apache-ARIES	Apache-TEZ	Jira-JSDSERVER	RedHat-ODC
RedHat-HAL	RedHat-HAC	Apache-HTTPCLIENT	Apache-ZEPPELIN	Apache-FOR	JiraEcosystem-PL
Apache-TRINIDAD	Apache-JCLOUDS	RedHat-SWITCHYARD	Apache-KYLIN	Apache-FOP	Jira-BSERV
Apache-GERONIMO	Jira-BAM	RedHat-RESTEASY	RedHat-TEIIDDES	Apache-BATIK	Apache-MSHARED
Apache-TOMAHAWK	RedHat-GITOPS	RedHat-CEQ	Apache-AVRO	Jira-CONFCLOUD	RedHat-WINDUP
Apache-OWB	JFrog-TCAP	Apache-AMQ	Apache-PARQUET	JiraEcosystem-AG	Apache-MRELEASE
Apache-TOBAGO	Apache-KARAF	Apache-AXIS2	Apache-CAY	Apache-ROL	Jira-I18N
RedHat-JBSEAM	RedHat-SB	RedHat-ENTMQMAAS	Apache-FLUME	Jira-JRASERVER	Spring-IDE
Apache-PIVOT	JFrog-RTFACT	RedHat-JBESB	Apache-PIG	Apache-JCR	RedHat-AF
RedHat-THORN	RedHat-OCPPLAN	RedHat-JBTM	Apache-HOP	Apache-XALANJ	Jira-CRUC
RedHat-JBPAPP	RedHat-MON	Apache-DIRMINA	Apache-ORC	Apache-OAK	Apache-GROOVY
Apache-NUTCH	RedHat-OSSM	Apache-SSHD	Apache-TAVERNA	Apache-UIMA	RedHat-JBDS
Apache-COCOON	Apache-SCB	RedHat-ENTESB	Apache-IGNITE	Apache-TIKA	JiraEcosystem-PLUG
Apache-JS2	RedHat-RFE	Apache-KNOX	Spring-DATACMNS	Apache-CMIS	Apache-MRM
Spring-SWF	Jira-OPSGENIE	Apache-AIRAVATA	Apache-SENTRY	Jira-CONFSERVER	RedHat-JBIDE
RedHat-JWS	RedHat-QUARKUS	RedHat-APIMAN	RedHat-ISPN	Apache-PDFBOX	Jira-BCLOUD
RedHat-JBAS	Jira-CLOUD	Apache-DIRSERVER	RedHat-MODE	Apache-XERCESJ	Apache-LOG4J2
Apache-TAP5	Apache-CLOUDSTACK	Apache-JUDDI	Apache-HDDS	Apache-STANBOL	JiraEcosystem-PLE
Apache-STR	RedHat-KATA	RedHat-EJBTHREE	Apache-GOBBLIN	Apache-JSPWIKI	RedHat-FUSETOOLS
Spring-SPR	RedHat-OPECO	Apache-DIRSTUDIO	Apache-OODT	Apache-LUCENE	Apache-NETBEANS
RedHat-JBEAP	RedHat-SDN	Apache-SYNAPSE	Apache-HUDI		Spring-STS
RedHat-WELD	Sonatype-NEXUS	RedHat-ENTMQBR	Apache-DRILL		JiraEcosystem-ACJIRA
Apache-RAVE	RedHat-SRVKP	RedHat-JGRP	Apache-HDFS		JiraEcosystem-AMPS
Apache-SLING	RedHat-HAWKULAR	Apache-OPENEJB	Apache-ACCUMULO		JiraEcosystem-FRGE
Apache-WW	RedHat-CLOUD	Apache-CXF	Apache-IOTDB		Apache-CONTINUUM
RedHat-DEVELOPER	JiraEcosystem-UPM	Spring-INT	Apache-FLINK		Jira-CLOV
Apache-BEEHIVE	RedHat-CNV	Apache-ARTEMIS	Apache-METRON		JiraEcosystem-AUI
Apache-MYFACES	Sonatype-MVNCENTRAL	Apache-AXIS	Spring-DATAMONGO		JiraEcosystem-APL
Apache-SHINDIG	Sonatype-OSSRH	Apache-TUSCANY	Apache-BOOKKEEPER		Apache-SUREFIRE
RedHat-WFLY	JFrog-HAP	RedHat-ENTMQST	Apache-YARN		Jira-FE
RedHat-AS7	Apache-ZOOKEEPER	Spring-SWS	Apache-PHOENIX		JiraEcosystem-AC
Apache-TAPESTRY	RedHat-RHCLOUD	Apache-OLINGO	Apache-FALCON		RedHat-ARQ
		Apache-CAMEL	Apache-BIGTOP		Apache-IVY
		RedHat-ENTMQ	Apache-SLIDER		RedHat-FORGE
			Apache-HIVE		
			Apache-CALCITE		
			Spring-DATAREDIS		
			Apache-GEODE		
			Apache-STORM		
			Spring-DATAES		
			Apache-LENS		
			Apache-HBASE		
			RedHat-DBZ		
			Spring-XD		
			Spring-DATAJPA		
			Apache-APEXMALHAR		
			Apache-ASTERIXDB		
			Apache-HADOOP		
			Spring-DATAGRAPH		
			Apache-DERBY		
			Apache-ATLAS		
			Apache-RANGER		
			MongoDB-JAVA		
			Apache-REEF		
			Apache-HAMA		
			Apache-TAJO		
			Spring-DATAREST		
			Apache-SQOOP		
			Apache-OPENJPA		
			RedHat-TEIID		
			Apache-EAGLE		
			Apache-MAPREDUCE		

C RQ3: Additional Results

In the report, only the simplified view of results in RQ3 was displayed, because the multiple dimensions of the data make for an enormous amount of results. Therefore, in this appendix, the full, domain-specific, intersected views can be found.

C.1 Chi-Squared Tables

All Domains	Exis	Exec	Prop	Non-Arch
Independent	0.93	0.99	0.96	1.01
Child	1.60		1.05	0.96
Parent	3.21	1.89	3.98	0.67
CM	Exis	Exec	Prop	Non-Arch
Independent	0.97	0.98	0.98	1.00
Child	1.93	1.66	1.38	0.90
Parent	4.09	3.01	4.94	0.63
DSP	Exis	Exec	Prop	Non-Arch
Independent	0.85	0.96	0.93	1.02
Child	1.69	1.05		0.94
Parent	3.23	2.47	4.03	0.60
DC	Exis	Exec	Prop	Non-Arch
Independent	0.98	1.03	1.01	0.99
Parent	2.77		2.33	0.79
Child		0.37	0.48	1.20
SOAM	Exis	Exec	Prop	Non-Arch
Independent	0.97	0.97	1.02	1.01
Child	1.15	1.22	0.54	0.97
Parent	2.36	1.94	1.67	0.76
SDT	Exis	Exec	Prop	Non-Arch
Parent	3.32	1.95	2.90	0.77
Independent	0.97	0.95		1.01
Child	1.16	1.83	0.51	0.91
WD	Exis	Exec	Prop	Non-Arch
Independent	0.97	0.96	0.98	1.01
Child	1.15	1.62		0.91
Parent	2.76	1.92	3.80	0.75

Table C.5: Chi-squared test results for the relation between the hierarchy issue characteristic and simple decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Prop	Non-Arch
Sub-Task	1.93	1.33	1.11	0.92
Bug	0.30	0.41	0.37	1.10
Improvement	1.87	1.35	2.42	0.89
New Feature	4.40	2.08	3.32	0.67
Wish	1.90	2.55	2.06	0.79
Task	1.11	3.12	0.88	0.81
CM	Exis	Exec	Prop	Non-Arch
Sub-Task	3.28	1.53	1.73	0.87
Bug	0.41	0.44	0.41	1.06
Improvement	2.60	1.94	3.35	0.85
New Feature	6.57	3.20	3.98	0.62
Wish	2.16	3.62	1.80	0.80
Task		6.33	1.24	0.70
DSP	Exis	Exec	Prop	Non-Arch
Bug	0.28	0.44	0.32	1.12
Task	1.10	2.66	0.78	0.85
Improvement	1.50	1.33	2.03	0.90
Sub-Task	1.75	1.09		0.93
New Feature	3.54	2.03	3.03	0.66
Wish	1.61	2.69	1.83	0.78
DC	Exis	Exec	Prop	Non-Arch
Task	1.31	2.36		0.86
Bug	0.32	0.49	0.48	1.10
Sub-Task	1.65	1.09	0.54	0.96
Improvement	1.96	1.42	2.48	0.88
New Feature	4.13	2.05	3.54	0.68
Wish		3.13	3.67	0.74
SOAM	Exis	Exec	Prop	Non-Arch
Improvement	1.89	1.15	2.18	0.91
New Feature	4.51	1.91	3.12	0.63
Bug	0.33	0.33	0.48	1.14
Task	0.87	2.56	0.81	0.81
Sub-Task	1.26	1.34	0.55	0.95
Wish	2.06	1.94	2.66	0.79
SDT	Exis	Exec	Prop	Non-Arch
Task	1.92	3.70	1.28	0.73
Bug	0.31	0.44	0.57	1.07
Sub-Task	1.79	2.11		0.88
New Feature	5.98	1.64	3.58	0.79
Wish	3.29	2.36	2.76	0.80
Improvement	2.50	1.52	2.83	0.90
WD	Exis	Exec	Prop	Non-Arch
Improvement	2.10	1.23	2.24	0.92
New Feature	5.09	1.91	3.29	0.71
Bug	0.29	0.41	0.39	1.09
Sub-Task	1.39	2.00		0.89
Task		3.12	1.17	0.81
Wish	2.23	2.18	2.06	0.83

Table C.6: Chi-squared test results for the relation between the issue_type issue characteristic and simple decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.92	0.99	0.98	0.97	0.92	1.03	0.95	1.01
Child	1.78		0.70		1.37	0.39	0.72	0.96
Parent	2.91	1.82	4.19	3.33	5.33	2.45	6.74	0.71
CM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.98	0.98	0.96	0.98	0.97	1.01	0.91	1.00
Child	1.94	1.74	1.91		1.86	0.09	3.35	0.91
Parent	3.62	2.74	9.64	5.52	5.87		16.22	0.68
DSP	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.84	0.96	0.89	0.97	0.89		0.88	1.02
Child	1.91	1.08		0.91	1.19	0.51		0.94
Parent	2.80	2.34	6.31	3.25	5.03	3.81	8.90	0.65
DC	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.96	1.03	1.04			1.04	1.03	0.99
Parent	3.17		1.68		3.87			0.81
Child	1.44	0.38	0.15	0.58	0.58	0.22	0.25	1.17
SOAM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.97	0.97	0.95			1.04		1.01
Child	1.25	1.26		0.55	0.64	0.23		0.96
Parent	2.26	1.89	4.26	1.64	1.89		3.18	0.78
SDT	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Parent	3.38	1.99	2.76	3.83	3.65			0.78
Independent	0.96	0.94				1.05		1.01
Child	1.23	1.91		0.44		0.16		0.90
WD	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Independent	0.98	0.96	0.97		0.96			1.01
Child	1.22	1.67				0.40		0.91
Parent	2.32	1.88	5.34	2.70	5.41	2.89	5.27	0.78

Table C.1: Chi-squared test results for the relation between the hierarchy issue characteristic and intersected decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Sub-Task	2.11	1.37	1.20		1.46	0.41		0.92
Bug	0.34	0.42	0.09	0.43	0.19	0.78	0.08	1.09
Improvement	1.80	1.35	1.56	2.78	2.54	1.60	1.99	0.91
New Feature	4.10	1.79	9.75	1.80	5.08	2.08	9.37	0.72
Wish	1.54	2.45	5.54	1.51	1.99	2.76	7.45	0.82
Task	1.18	3.32		0.82	0.81	1.21	0.64	0.81

CM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Sub-Task	3.45	1.55	2.88	1.63	2.78	0.10	3.88	0.89
Bug	0.49	0.44	0.05	0.42	0.15	0.84	0.02	1.05
Improvement	2.43	1.97	2.42	3.94	3.98	1.60	3.18	0.87
New Feature	5.99	2.81	15.98	1.74	8.21	1.84	14.10	0.68
Wish	1.88	3.75	5.01				7.07	0.81
Task	0.78	6.94	2.02			1.96		0.70

DSP	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Bug	0.32	0.45	0.12	0.42	0.21	0.57	0.08	1.11
Task	1.22	2.85		0.77	0.71		0.55	0.85
Improvement	1.41	1.33	1.34	2.26	2.01	1.79	1.55	0.92
Sub-Task	1.98	1.12			1.22	0.52		0.93
New Feature	3.26	1.73	7.91	1.61	4.27	2.51	8.73	0.71
Wish		2.56	5.81		1.88	2.99	5.81	0.82

DC	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Task	1.30	2.46	1.99			1.34		0.86
Bug	0.36	0.48	0.09	0.46	0.14		0.04	1.09
Sub-Task	1.84	1.15		0.59	0.73	0.20		0.96
Improvement	1.89	1.46		2.99	3.17			0.89
New Feature	3.75	1.84	8.94	2.75	5.97	1.58	9.41	0.72
Wish		3.14		5.74		6.52		0.75

SOAM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Improvement	1.94	1.15	1.26	2.65	2.21	1.43	1.74	0.91
New Feature	4.12	1.59	10.49	1.74	4.97	1.96	8.18	0.68
Bug	0.37	0.32	0.04	0.54	0.30	0.83	0.19	1.12
Task	0.92	2.71	0.54	0.74	0.79		0.39	0.81
Sub-Task	1.35	1.38		0.55	0.69	0.20		0.94
Wish	1.66	1.85	4.33		3.12		7.27	0.82

SDT	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Task	1.96	3.83	1.70		1.78			0.73
Bug	0.33	0.43	0.14	0.50	0.21		0.12	1.07
Sub-Task	1.89	2.18				0.20		0.88
New Feature	5.70	1.48	10.22	2.84	6.99		9.37	0.81
Wish	2.77	2.19	10.12			3.85		0.82
Improvement	2.45	1.52	2.53	3.76	3.25		3.41	0.90

WD	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Improvement	2.13	1.21	1.65	2.56	2.29	1.49	2.13	0.93
New Feature	4.97	1.71	9.44	1.50	5.37	2.11	7.82	0.75
Bug	0.32	0.41	0.11	0.46	0.21	0.77	0.08	1.08
Sub-Task	1.46	2.07						0.89
Task		3.25			1.30			0.81
Wish	1.94	2.01	6.01				19.93	0.86

Table C.2: Chi-squared test results for the relation between the issue_type issue characteristic and intersected decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Won'T Fix	1.38	0.55	0.87	1.21	1.48	0.82	1.46	1.04
Duplicate	1.07	0.79		1.16	1.31			1.02
Fixed	0.94	1.03	1.16	1.03	0.98	1.06	1.07	1.00
Obsolete	0.77	0.51	0.44	0.52	0.39	0.49		1.09
Not A Bug	0.35	0.35	0.14	0.55	0.20			1.13
Done	1.06	1.22	0.58	0.78	0.82	0.87	0.62	0.97

CM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Won'T Fix	1.49	0.58	1.23	0.78	1.25	0.59		1.01
Duplicate	1.12	0.57	0.51			0.67		1.03
Fixed	0.84	1.27		1.17		1.28		0.99
Obsolete	0.39	0.65	0.38	0.36	0.16	0.17		1.06
Not A Bug	0.52	0.34			0.15			1.07
Done	1.39	2.84	2.75					0.86

DSP	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Fixed	0.94	0.96	0.92	0.95	0.89	0.94	0.82	1.01
Won'T Fix	1.52	1.13	2.27	1.65	2.22	1.45	3.01	0.93
Duplicate		0.91		1.29	1.40	1.73		
Done	1.27	1.35						0.96
Not A Bug	0.16	0.52		0.24	0.09			1.11
Obsolete								

DC	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Done	1.54	0.51	0.35	1.52	1.38	1.35	0.55	1.14
Duplicate	1.18	0.93						
Not A Bug	0.22	0.22	0.13					1.34
Obsolete	1.69	0.29	0.28		1.72			1.21
Fixed	0.83	1.19	1.22	0.78	0.85		1.19	0.94
Won'T Fix	1.25	0.48	0.37	2.22	1.52	0.39		1.17

SOAM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Won'T Fix	1.42	0.79	2.42		1.54	1.73	2.79	
Fixed	1.03	1.02	1.13			0.87		0.99
Duplicate	1.27					2.79		0.97
Done	0.88		0.58					1.01
Obsolete		1.39						
Not A Bug		0.42						1.10

SDT	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Done	1.25	1.50	1.29	0.56	0.77	0.36		0.94
Obsolete		0.52						1.05
Fixed	0.68	0.81	0.78			1.64		1.03
Won'T Fix	1.38	0.72		1.88	1.73	0.58	3.21	1.01
Duplicate	1.27	0.56		1.69	1.91			1.04
Not A Bug	0.45	0.54						1.08

WD	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Won'T Fix	1.64	0.53	1.72		1.81			1.03
Fixed	1.06	0.70						1.04
Duplicate	0.80	0.59						1.06
Done	0.84	1.44	0.76		0.78		0.49	0.95
Obsolete		1.94						0.89
Not A Bug		0.50				12.30		1.08

Table C.3: Chi-squared test results for the relation between the resolution issue characteristic and intersected decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Closed	0.89	1.08	1.05	0.83	0.74	0.88	0.90	
Open	1.45	0.70	1.32	1.82	2.21	1.46	2.03	0.99
Resolved	1.08	0.92	0.70	1.11	1.18	1.10	0.80	1.01
In Progress	2.59		3.46	2.21	3.17	2.07	3.26	0.88
Reopened	1.31	0.61						1.02

CM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Closed	0.97	0.91	0.79	0.85	0.82	0.85	0.75	1.01
Open	1.60	1.29	2.41	2.23	2.82	1.64	2.49	0.94
Resolved	0.92	1.69	1.87	1.61	1.55	2.01	2.41	0.95
In Progress	1.76	1.72	12.74					0.88
Reopened								0.95

DSP	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Resolved		0.97	0.83	0.89	0.86		0.72	1.01
Open	1.26	0.96	1.71	1.45	1.67	1.42	1.94	0.97
In Progress	1.61	1.25	3.78	1.55	2.20	1.96	2.73	0.90
Closed	0.88	1.05	0.84	0.92	0.85	0.75		1.01
Reopened	1.27							

DC	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Closed	0.94	1.08	1.13	0.86	0.81	0.96	1.09	0.98
In Progress	4.68	0.38	1.54	4.35	5.58			0.91
Resolved		0.84	0.25		1.33		0.40	1.07
Open	1.45	0.31	0.38	2.55	2.60	1.44		1.18
Reopened		0.34						1.18

SOAM	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Closed	0.97	0.84	0.80	0.93	0.88		0.81	1.03
Resolved		1.32						0.96
Open	1.36	0.78	2.30	1.65	1.90	1.59	3.72	0.98
Reopened		0.45						1.07
In Progress	2.62	1.91	3.92					0.75

SDT	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Closed	0.93	1.02	0.86		0.94		0.73	1.00
Reopened	1.69							
Open	1.45	0.78	1.99	1.35	1.43		3.04	
Resolved	1.28		1.52					0.99
In Progress	1.83				5.04			0.94

WD	Exis	Exec	Exis-Exec	Prop	Exis-Prop	Exec-Prop	All	Non-Arch
Resolved	0.91	0.90						1.02
Closed	0.95	1.07	0.89		0.94	0.82		1.00
Open	1.80	0.69	2.42	1.59	1.93	1.96	3.11	0.99
In Progress	2.41	1.62				6.48		0.82
Reopened		0.48						1.05

Table C.4: Chi-squared test results for the relation between the status issue characteristic and intersected decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Prop	Non-Arch
Won'T Fix	1.33	0.58	1.27	1.03
Duplicate	1.09	0.81	1.21	1.02
Fixed	0.97	1.04	1.02	1.00
Obsolete	0.69	0.51	0.46	1.10
Not A Bug	0.31	0.36	0.44	1.15
Done	0.98	1.18	0.81	0.98

CM	Exis	Exec	Prop	Non-Arch
Won'T Fix	1.44	0.61	0.92	1.01
Duplicate	1.09	0.57		1.03
Fixed	0.87	1.26	1.11	0.99
Obsolete	0.36	0.61	0.24	1.07
Not A Bug	0.44	0.34	0.46	1.08
Done	1.43	2.76		0.85

DSP	Exis	Exec	Prop	Non-Arch
Fixed	0.93	0.96	0.91	1.01
Won'T Fix	1.67	1.19	1.91	0.91
Duplicate	1.09	0.93	1.39	0.99
Done	1.23	1.32		0.95
Not A Bug	0.14	0.50	0.22	1.13
Obsolete				

DC	Exis	Exec	Prop	Non-Arch
Done	1.15	0.52	1.31	1.16
Duplicate	1.14	0.94		
Not A Bug	0.19	0.22	0.34	1.37
Obsolete	1.23	0.30		1.23
Fixed	0.96	1.18	0.91	0.94
Won'T Fix		0.48	1.30	1.19

SOAM	Exis	Exec	Prop	Non-Arch
Won'T Fix	1.52	0.88	1.40	0.98
Fixed	1.03	1.03		0.99
Duplicate	1.25		1.53	0.96
Done	0.86	0.97	0.92	1.02
Obsolete		1.36		
Not A Bug		0.44		1.10

SDT	Exis	Exec	Prop	Non-Arch
Done	1.21	1.47	0.58	0.94
Obsolete		0.54		1.05
Fixed	0.70	0.83	1.14	1.03
Won'T Fix	1.40	0.73	1.50	1.01
Duplicate	1.27	0.56	1.51	1.04
Not A Bug	0.45	0.54		1.08

WD	Exis	Exec	Prop	Non-Arch
Won'T Fix	1.65	0.57	1.37	1.02
Fixed	1.06	0.72		1.04
Duplicate		0.61		1.06
Done	0.83	1.41	0.87	0.95
Obsolete		1.90		0.90
Not A Bug	0.31		3.08	

Table C.7: Chi-squared test results for the relation between the resolution issue characteristic and simple decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

All Domains	Exis	Exec	Prop	Non-Arch
Closed	0.89	1.07	0.80	1.00
Open	1.52	0.75	1.92	0.98
Resolved	1.05	0.91	1.12	1.01
In Progress	2.68	1.11	2.56	0.84
Reopened	1.28	0.66	1.32	1.02

CM	Exis	Exec	Prop	Non-Arch
Closed	0.94	0.90	0.84	1.01
Open	1.78	1.35	2.30	0.92
Resolved	1.07	1.71	1.68	0.94
In Progress	2.37	2.25		0.85
Reopened			2.28	0.94

DSP	Exis	Exec	Prop	Non-Arch
Resolved	0.96	0.96	0.89	1.01
Open	1.36		1.55	0.95
In Progress	1.83	1.40	1.90	0.87
Closed	0.88	1.03	0.87	1.01
Reopened	1.28			0.97

DC	Exis	Exec	Prop	Non-Arch
Closed	0.98	1.08	0.89	0.98
In Progress	3.68	0.48	3.76	0.87
Resolved	0.81	0.81		1.08
Open	1.22	0.33	2.07	1.18
Reopened		0.46		1.16

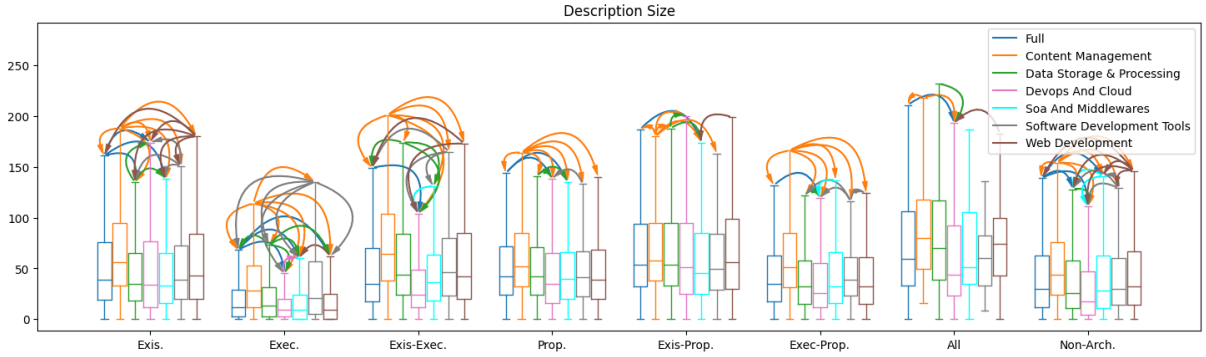
SOAM	Exis	Exec	Prop	Non-Arch
Closed	0.94	0.84	0.90	1.03
Resolved		1.30		0.96
Open	1.52	0.88	1.81	0.96
Reopened		0.45		1.08
In Progress	2.63	1.94		0.73

SDT	Exis	Exec	Prop	Non-Arch
Closed	0.92	1.02	0.97	1.00
Reopened	1.67			
Open	1.49	0.82	1.31	
Resolved	1.29			0.99
In Progress	1.98		2.52	0.93

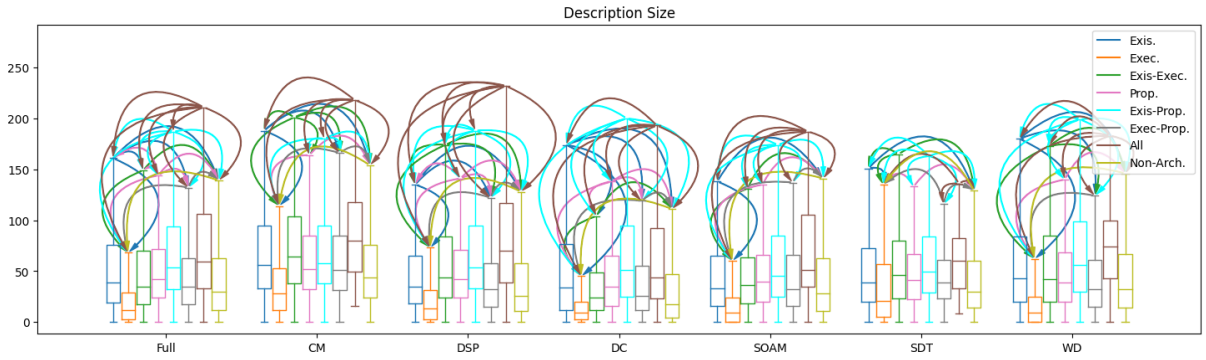
WD	Exis	Exec	Prop	Non-Arch
Resolved	0.90	0.91		1.02
Closed	0.94	1.06	0.93	1.00
Open	1.85	0.77	1.83	0.97
In Progress	2.59	1.76	3.28	0.78
Reopened		0.54		1.05

Table C.8: Chi-squared test results for the relation between the status issue characteristic and simple decision type contained in the issue, separated by domain. (CM = Content Management, DSP = Data Storage and Processing, DC = DevOps and Cloud, SOAM = SOA and Middlewares, SDT = Software Development Tools, WD = Web Development)

C.2 Mann-Whitney Plots

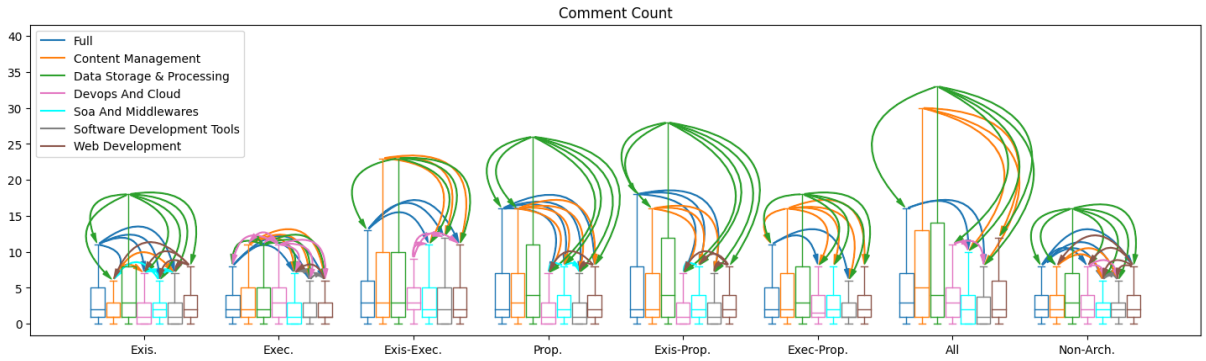


(a) Grouped by decision type first and subgrouped by domain.

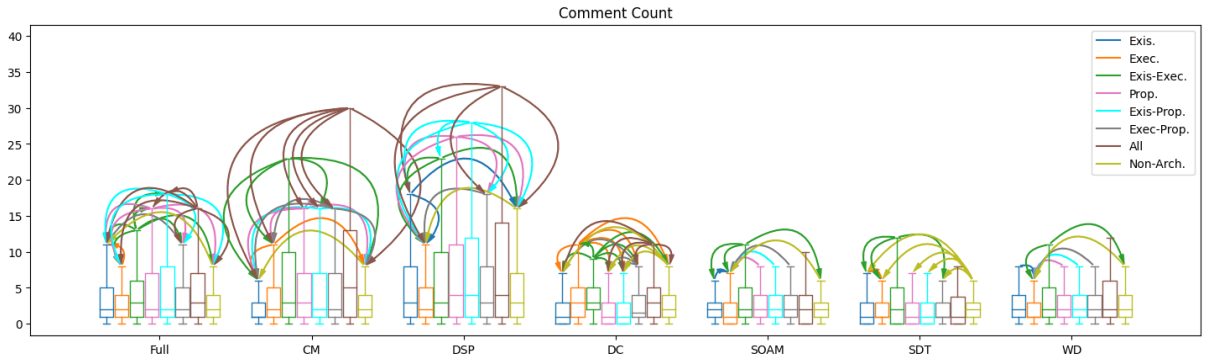


(b) Grouped by domain first and subgrouped by decision type.

Figure C.1: Mann-Whitney test on the relations between description size, domain and intersected decision type contained in the issue.

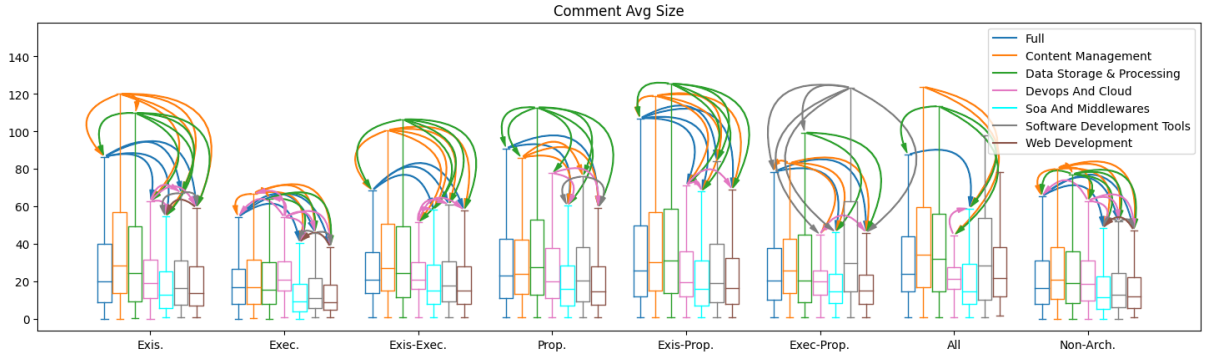


(a) Grouped by decision type first and subgrouped by domain.

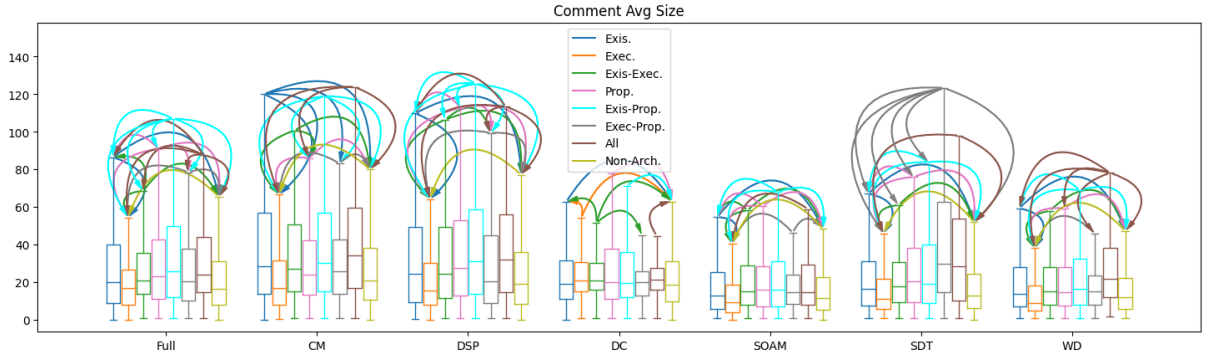


(b) Grouped by domain first and subgrouped by decision type.

Figure C.2: Mann-Whitney test on the relations between comment count, domain and intersected decision type contained in the issue.

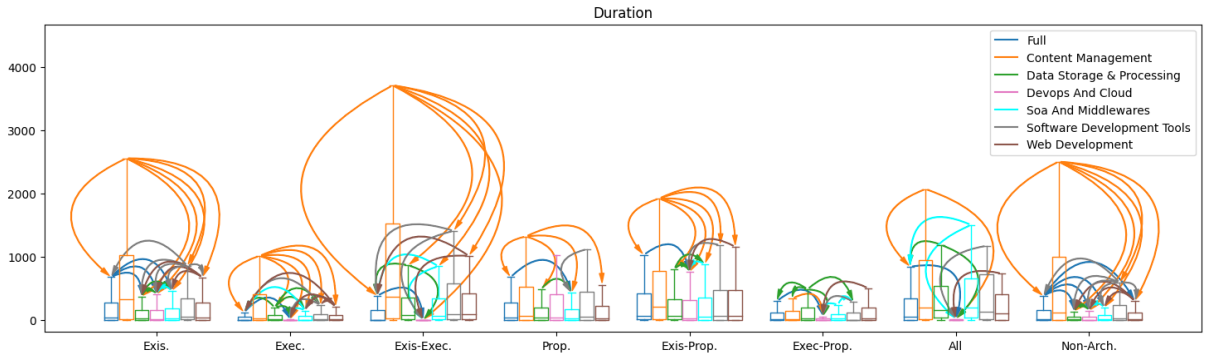


(a) Grouped by decision type first and subgrouped by domain.

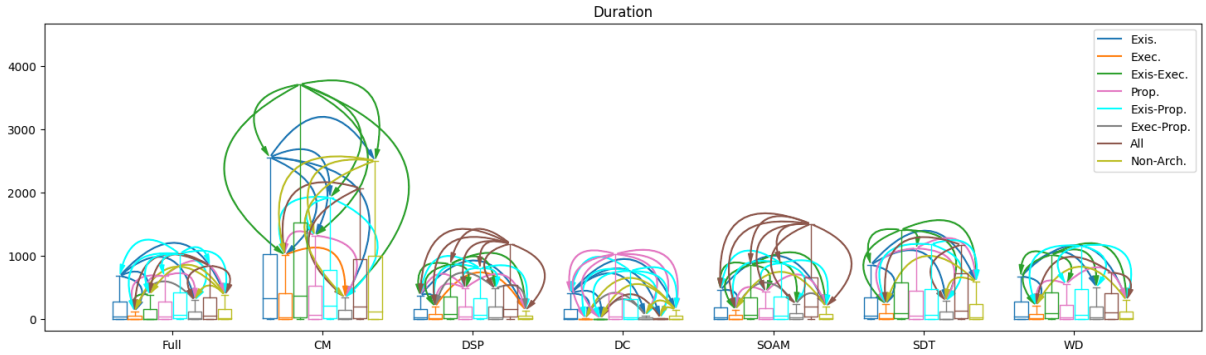


(b) Grouped by domain first and subgrouped by decision type.

Figure C.3: Mann-Whitney test on the relations between comment average size, domain and intersected decision type contained in the issue.

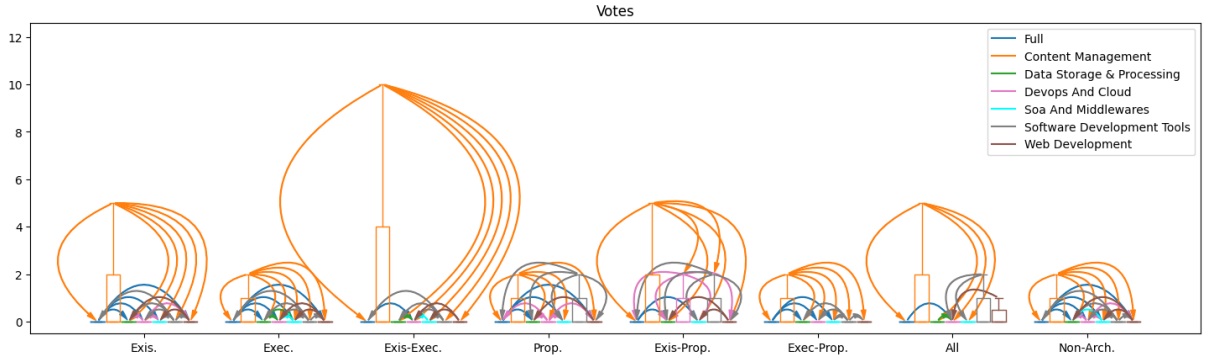


(a) Grouped by decision type first and subgrouped by domain.

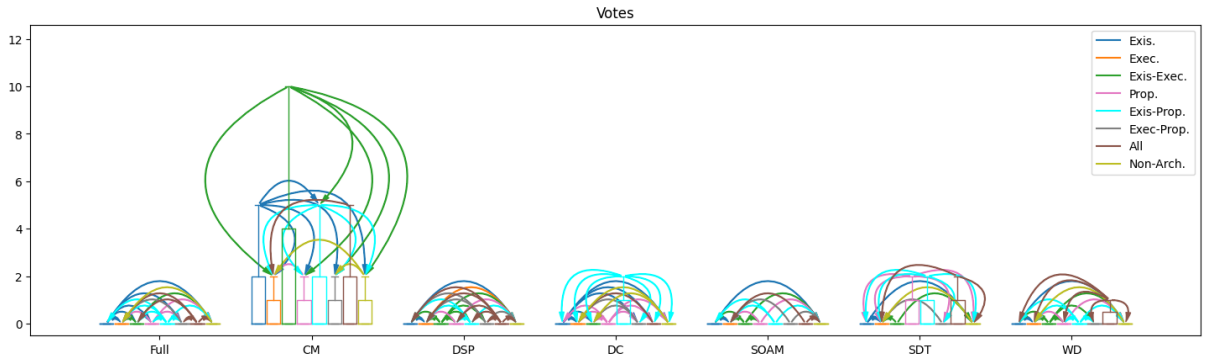


(b) Grouped by domain first and subgrouped by decision type.

Figure C.4: Mann-Whitney test on the relations between duration, domain and intersected decision type contained in the issue.

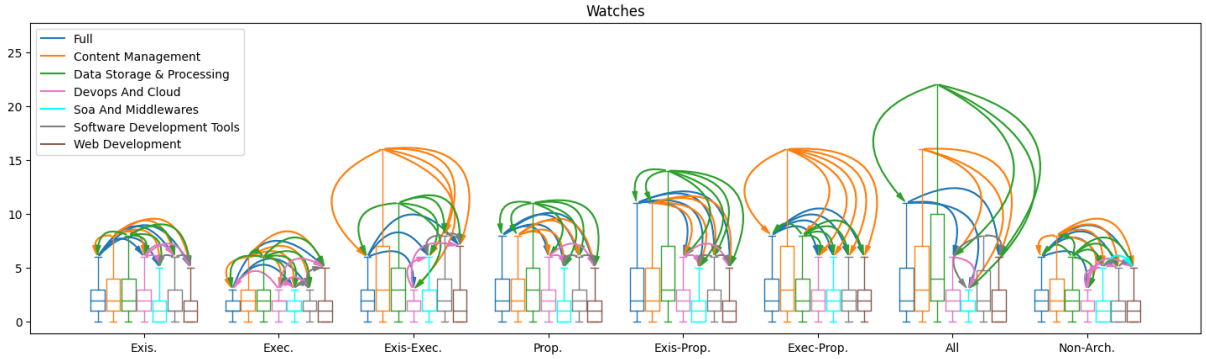


(a) Grouped by decision type first and subgrouped by domain.

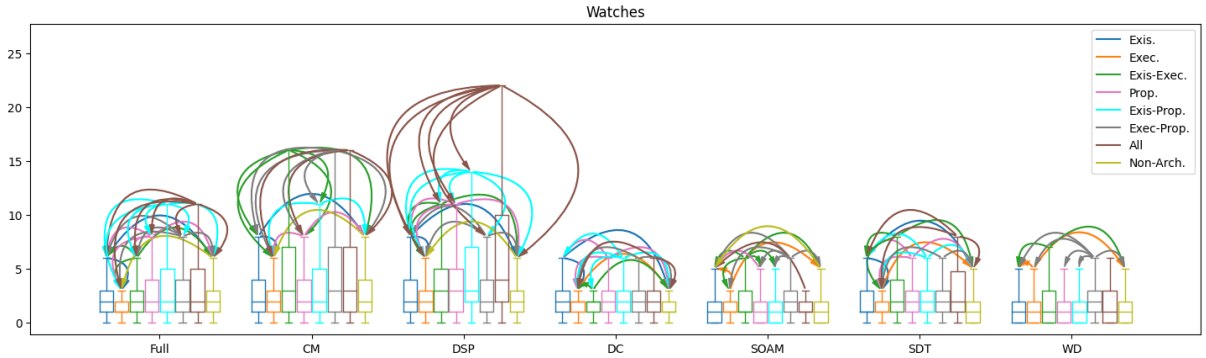


(b) Grouped by domain first and subgrouped by decision type.

Figure C.5: Mann-Whitney test on the relations between votes, domain and intersected decision type contained in the issue.



(a) Grouped by decision type first and subgrouped by domain.



(b) Grouped by domain first and subgrouped by decision type.

Figure C.6: Mann-Whitney test on the relations between watches, domain and intersected decision type contained in the issue.