Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

Master's thesis

Investigating JackTrip's Software Architecture

A Case Study On Prominent Reasons For Architectural Erosion In Open Source Software Originating From Academia

For obtaining the academic degree Master of Science Media Informatics

Submitted by: Sven Thielen
Matriculation number: [redacted]

Degree program: M.Sc. Media Informatics
Exam regulations: [redacted]

First examiner: Prof. Thomas Franz, PhD
Second examiner: Prof. Chris Chafe, DMA

Submission date: 31.05.2023

This page intentionally left blank.

Abstract

Sustainable software architecture is crucial for the success of an evolving system that is affected by its context and environment. However, deviations between software design and implementation are often uncontrolled and induce architectural erosion. The literature describes several symptoms and observes this phenomenon in open source software. Prominent reasons for the occurrence of architectural erosion include a rushed evolution, recurring changes, a lack of developers' awareness, time pressures, and the accumulation of design decisions. Architectural erosion is also prevalent in academic projects, but it still needs to be researched in-depth in a real-life setting.

In this work, a single case study investigates the symptoms and reasons for architectural erosion in open source software originating from academia by employing a mixed-methods approach. Architectural metrics from a static code analysis using ARCADE Core are contrasted with developers' perspectives sourced from semistructured interviews. The investigated project is JackTrip, a cross-platform application for realtime networked music performances and Internet reverberation research.

The first research question aims at assessing the trend of structural symptoms for architectural erosion in JackTrip. The results show a significant increase in link overload smells in the course of the evolution, which reached 11 as of version 1.6.2. This supports the developers' perception of a rushed evolution corresponding to the second research question, which asks to what extent prominent causes for architectural erosion apply. Additionally, the interview data evince time constraints related to tight research funding and the commercialization of the project. Research question 3 investigates how the academic environment and the problem domain relate to the symptoms and reasons for architecture erosion. It has been found that these influences shape the development approach and impact contributor selection.

In relation to a certain inexperience and lack of commitment of open source developers as a reason for architectural erosion, the results suggest further studies and opt for ending the discussion regarding skills. Currently, the role of an architect or dedicated architectural work does not seem to be present in the JackTrip project. However, structural issues described in this work should not be considered as categorical statements about its software architecture as they would need to be confirmed on a code level.

Acknowledgements

I would like to explicitly thank some colleagues and researchers who accompanied me during the process of writing. First, I am glad to be eligible to explore the inner workings, initiators and community driving the JackTrip project. I sincerely thank Professor [redacted] at the Center for Computer Research in Music and Acoustics of Stanford University, who willingly let me investigate JackTrip in obtaining the academic degree Master of Science Media Informatics.

Second, I am grateful being allowed to use the facilities at Hochschule Düsseldorf University of Applied Sciences. The utterly tranquil atmosphere at the networking laboratory allowed me to dive deeply into some topics without distraction, and the tools on site provided a desirable working place. The person taking care of that clean architecture is [redacted], who is supposed to be honored by this reference to *Uncle Bob*.

Talking with colleagues at the Institute of Sound and Vibration Engineering (ISAVE) supported me to remain focused and preserve confidence during my research. To name a few, I would like to say thank [redacted] and Professor [redacted]. I would also like to thank Professor [redacted], a great mentor and cool boss of that team, for governing the faculty of media.

The advanced measurements and their interpretation would probably have been much more opaque without the comprehensible ARCADE open source software, which is currently rewritten as ARCADE Core by Marcelo Schmitt Laser and was initially created by the Software Architecture Research Group at the University of Southern California. Feature and bug reports as well as discussions on the findings have been beneficial for both parties. I admire Marcelo's broad knowledge and accumulated experience in software architecture and system evolution.

Another important resource has been the autonomous time management that has been granted to me with confidence by Professors [redacted] and [redacted] at Hochschule Niederrhein University of Applied Sciences, despite my being a full-time research assistant at the Cyber Security Management Campus Mönchengladbach. Last but not at least, I offer many thanks to Professor [redacted] at Hochschule Düsseldorf University of Applied Sciences for supervising this thesis by directing me in the planning phase, pointing to literature, and reviewing changes throughout the evolution of this architecture of words.

Table of Contents

1 Introduction

It is well known that during the lifecycle of a software project, an on-going process of decay forces a system to adapt to changing requirements and environmental conditions. By doing so, software systems become more complex over time (Whiting & Andrews, 2020). As a consequence, they need to be designed and implemented with regard to the principles of extensibility, maintainability, and sustainability (Koschel et al., 2019).

1.1 Overview of the Subject

Software architecture (SA) addresses the aforementioned requirements by continuously defining, documenting, maintaining and improving, the “fundamental concepts or properties of a system in its environment” (ISO, 2011). While the implementation of a software system evolves, its SA design needs to be aligned, and vice versa. Otherwise, the actual architecture deviates from the intended one and can render a system unmaintainable over time. This process is known as architectural erosion (AE) and may cause low performance, new bugs due to refactoring, and a state where a software system is hardly understood, particularly by on-boarding developers (Li et al., 2021).

1.2 State of the Art in Research

Li et al. (2022) reviewed studies on the phenomenon of AE from January 2006 to May 2019 in a systematic mapping study for analysis and categorization, especially identify reasons, symptoms, and consequences of AE. They propose evaluating and quantifying AE symptoms at a structural level, such as architectural smell (AS) occurrences, using existing tools, and they claim the lack of empirical case studies exploring the reasons and consequences of AE in a real-life setting.

1.3 Problem Statement

Since the last decade, the use of open source software (OSS) in commercial applications has increased, which makes investigating AE in this field an objective of current software research. Accoding to Baabad et al. (2020), most OSS projects face AE because of one or more of the following five most prominent causes:

i. Rushed evolution,

ii. Recurring changes,

iii. Lack of developers' awareness,

iv. Time pressure, and

v. Accumulation of design decisions.

The origin of these causes might be investigated in a more in-depth study, for example a case study. Employing static code analysis to measure AS occurrences as a symptom of AE using existing tools for software evolution research, such as the ARCADE software, is proposed by Baabad et al. (2020). In addition, interviews with developers may support in interpretation of the findings, provide contextual information such as individual reasons for engagement (Coelho et al., 2018), and trace AE to design decisions in the evolution of the project. Concerning environmental conditions Laser et al. (2021) observe evolutionary peculiarities in academic and research OSS comparable to archipelagos and suggest to take that aspect into account when evaluating such systems.

1.4 Research Questions

Given that the project of interest for a case study is an OSS project that originates from academia and faces AE, the following research questions (RQ_i) are derived from the problem statement and guide the investigations:

- **RQ_1:** What is the trend of structural symptoms for AE in OSS originating from academia in a contemporary project assessed by a static code analysis?
- **RQ_2:** To what extent do the prominent causes of AE in OSS apply to the chosen academic OSS project?
- **RQ_3:** How do the academic environment and problem domain relate to the symptoms and reasons for AE in that OSS project?

Following the argument that OSS projects often attract people contributing in their spare time (Baabad et al., 2020) and assuming that the academic environment urges one to focus on the problem domain while neglecting extensible and maintainable structures (Zirkelbach et al., 2019), three research hypotheses (RH_i) can be drawn:

- **RH_1:** The chosen OSS academic project might unnoticeably accumulate an AS number that keeps increasing during its evolution.
- **RH_2:** In an academic OSS project, rushed evolution, recurring changes and time pressure may occur less than in professional environments, making other reasons for AE more plausible.
- **RH_3:** Domain-specific circumstances and the academic environment shape the development approach in the OSS project and impact contributor selection.

1.5 Motivation and Relevance

A study on a currently fast-evolving OSS project in academia that may experience symptoms of AE may be a suitable case to investigate the potential reasons. The value of such findings would be a more comprehensive understanding of the phenomenon and its consequences in a real-life setting. Additionally, a case study could illustrate the influences of academia and take further environmental aspects into account: project context, organizational structures, process models, or even the problem domain itself.

1.6 Structure of This Thesis

Beginning with an introduction, the first chapter gives an overview of the subject and describes the state of the art for defining the problem statement. Using that base, the research questions and the motivation and relevance are drawn before explaining the structure of this thesis in subchapter 1.6. A theoretical backround is established in Chapter 2. After characteristics of open source software evolution and software architecture fundamentals are introduced, subchapter 2.3 deals with the details of analysis and recovery of software architecture followed by symptoms and reasons for architectural erosion and theoretical concepts of architecture recovery techniques and architectural metrics. After describing the methodology of this case study in Chapter 3 by explaining the used approaches and tools, the results are presented and discussed in Chapter 4 and 5. A conclusion and outlook reflects on the findings of this work and proposes further research directions followed by bibliography, further attachments such as a list of abbreviations and interview transcripts in the appendix, and an obligatory declaration on oath.

2 Theoretical Background

In this chapter, the evolution of open source and academic projects is described. Further, software architecture fundamentals and a background on the analysis and recovery of SA complete this theoretical chapter to create a foundational starting point.

2.1 Open Source Software Evolution

A software project typically starts with a personal problem. The main motivation might be the problem domain itself or a missing feature in an existing solution. Open source software (OSS) differs from closed source or proprietary software in that the source code is publicly available, inviting other people to contribute. The chosen license allows source code sharing and empowers users to change the software they use.

2.1.1 Open Source Definition

Referring to The Open Source Definition (OSD) compiled by the Open Source Initiative (2007), further details of redistribution, deriving, and antidiscrimination characterize OSS. The rationale for allowing derivations or forks of a project is to support rapid evolution by easing modification and its redistribution. Conversely, the original authors and maintainers cannot be held responsible for unofficial changes. Thus, those modifications must be distinguishable from the base to ensure the integrity of the author's source code and protect the reputation of maintainers regarding their scope of support.

To prevent discrimination against persons, groups, and fields of endeavor, there are conditions that allow a diverse community and include commercial use cases. Indirect means of closing up the software, for example by a non-disclosure agreement, or restricting the software to a specific product, are intercepted by specific clauses. Lastly, other software, that is distributed along with it, cannot be obliged to use the same license, regardless of whether it is distributed via the same medium. Additionally, the license must be neutral to technology in order to foster code reuse independent of specific platforms and interfaces. The entirety of these clauses reflects the values of a code-sharing culture and have been driven by practitioners such as Eric S. Raymond (Open Source Initiative, 2007).

2.1.2 The Fetchmail Example

An example for the evolution of open source software is the Fetchmail[1] project, which was begun in 1993 by Eric S. Raymond, who was looking for a POP3 client that would automatically set the correct reply address after mail is fetched to his computer at home. Instead of creating a completely new software, he contributed to the Fetchpop project by sending necessary code adaptions to the author who accepted them in a subsequent release. He provided some more patches and became familiar with the design of Fetchpop. Three years later, he decided to switch to another codebase, as Popclient was more appealing to him. The original author had already lost interest in it, so Raymond took over the project and became its new maintainer:

> In a software culture that encourages code-sharing, this is a natural way for a project to evolve (Raymond, 2000).

By steadily recruiting betatesters from the existing user base, finding and solving issues was faster as debugging could be shared among the community. The main difference to what Raymond (2000) calls the “cathedral-style” typical for proprietary software is the attitude of actively including users into the development process and working as cooperatively as possible. This welcoming culture attracted people, and upcoming technology such as the Internet and version control systems allowed distributed development.

With a growing user base and onboarding co-developers, communicating to identify promising ideas for implementation while trying to keep the code base clean and looking for a streamlined application became challenging. An example of this was when Raymond received a code proposal allowing him to forward fetched mail via the simple mail transport protocol. He decided to make that way of delivery the default mode, dropped support for other modes, and renamed the project to Fetchmail; it focused on mail transfer agent capabilities, only and soon became a musthave.

The social characteristic of OSS addresses building communities of people around shared interests and collaboratively working on code. The code-sharing culture, as Raymond (2000) describes it, ensures the continuous lifecycle of the software. Communities can evolve a system and split up to form a new project if conflicting objectives arise.

1 https://www.fetchmail.info

2.1.3 Academic Projects

Creating software in academic contexts is mainly driven by the objective to produce quantitatively measurable data by algorithmically implementing a theoretical model (de Souza et al., 2019), but this does not necessarily account for professional software engineering practices, as used in commercial settings according to Groen et al., (2015). Although the latter supports research transparency, for example by allowing reproducible deployment, having to devote time to both activities does not appeal to researchers. Groen et al. found that researchers learn software development through self-study or by talking to peers rather than through formal training. Additionally, scientists rate the need for software engineering practices usually more important in larger software projects and share the attitude that academic software is not built for comprehensibility, maintainability, and extensibility, but primarily for achieving accurate results and stability.

On the contrary, Groen et al. (2015) show that other investigated projects incorporate software engineering practices in general, but team size and transiency force a looser application. Laser et al. (2021) state that the principal drivers of academic software are graduate students and post-doctoral researchers who contribute temporarily to advance a topic of interest without concern for the characteristics of a mature software tool. A reason for that focus is that authors of research software do not expect it to be used by people other than themselves, as described in the following:

> In the academic world, small software proofs-of-concept (POC) are most frequently developed by researchers to run experiments that tend not to generalize. POC are usually discarded shortly after they fulfill their purpose (e.g., when a submitted paper is accepted, or M.S. thesis or Ph.D. dissertation completed). However, there are cases when research-based software grows into large, multi-purpose components, tools, frameworks, workbenches, and/or environments (e.g., when the developed capabilities are perceived by a new Ph.D. student as a good foundation for their own dissertation research). (Laser et al., 2021)

Additionally, organizational structures in academia insist that new ideas be published on a daily basis while limiting resources by tight funding, which encourages production os so-called “throwaway code” as the complexity of the resulting software is not evaluated most of the time (Laser et al., 2021). A software design and implementation geared towards sustainability is a fundamental framework to mitigating code dumping.

2.2 Software Architecture Fundamentals

To understand how sustainable software systems can be created, the term *software architecture* (SA) needs to be defined. There are different definitions in the literature, and the software industry has not yet consented to a specific one. Therefore, SA definitions range from early statements by practitioners to the approaches of the Institute of Electrical and Electronics Engineers for creating a standard. In 1992, Dewayne Perry and Alex Wolf described SA as follows:

> A set of architectural (or [...] design) elements that have a particular form.

Other historical and bibliographical definitions can be found in a list compiled by the Carnegie Mellon University Software Engineering Institute (2010). It also provides IEEE's first consensus on formally defining SA with standard 1471-2000, which has been superseded by the International Organization for Standardization's (ISO) standard 42010:2011, Systems and Software engineering – Architecture Description (ISO, 2011).

A common ground and classical theory is that SA is about partitioning a complex software system into smaller logical parts called *building blocks*, also known as *components* or *modules*, to meet certain quality requirements (QRs) such as maintainability. However, some real-life problems to be solved by software projects are naturally difficult to decompose, and with that maintenance-only point of view, a SA cannot account for ecological, legal, social, and other nontechnical influences on a software system that constitute its environment. For that reason, the following definition outlines the key elements that most appropriately and precisely illustrate their intersections:

> The software architecture defines the fundamental principles and rules for the organization of a system and its structure into building blocks and interfaces, and their relationships to each other and to the surrounding environment. It thus defines guidelines for the entire software lifecycle, the developer, and the software's operator, from analysis via design and implementation to operation and enhancement. (Koschel et al., 2019)

This definition explicitly takes context into account. Additionally, Koschel et al. (2019) point out the relation between structural and procedural aspects and emphasize their nature of being subject to change, thus making the structure, process, and developers interdependent throughout the lifecycle of a software system.

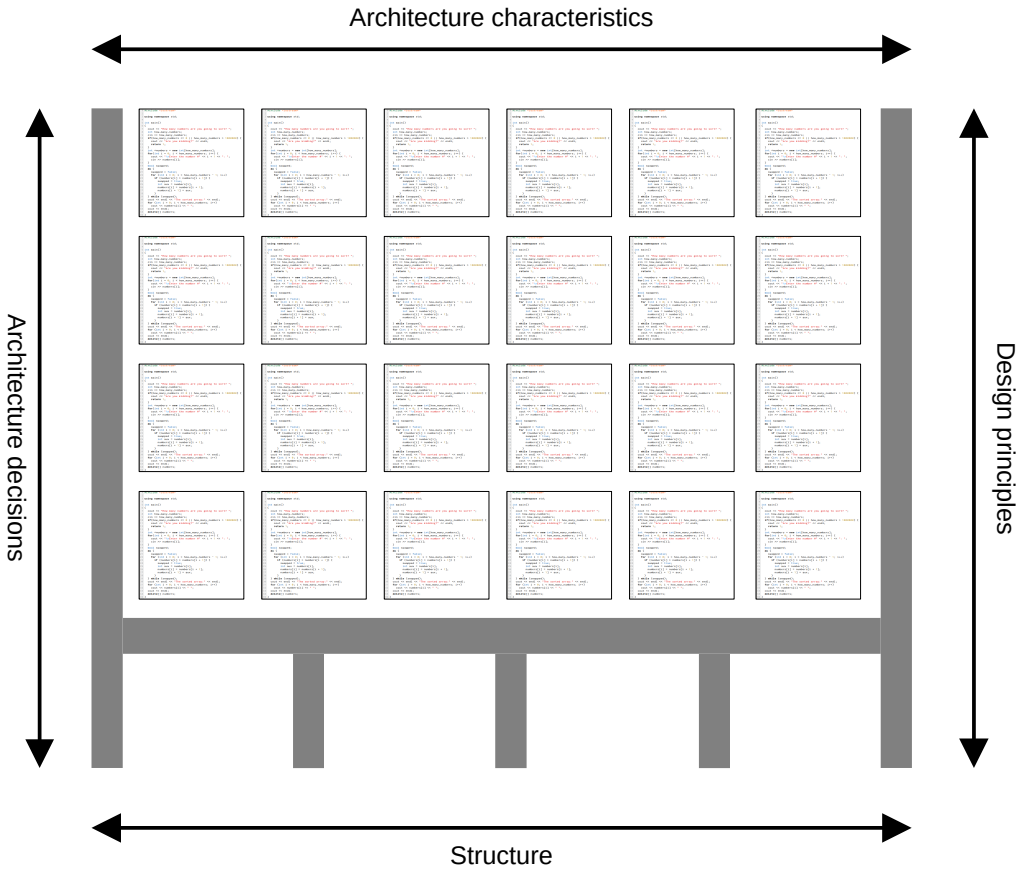

Figure 1: Four dimensions of software architecture (Richards & Ford, 2020)

Apart from that, Koschel et al. describe the concept of views depending on the party of interest, also known as a *stakeholder*, and their documentation in a collection of artifacts (Koschel et al., 2019). To begin looking at SA from a structural viewpoint, it is like bare brickwork carrying the source code that is supported by the following surrounding factors as illustrated in Figure 1, according to Richards and Ford (2020):

1. Structure (or Architectural Styles),
2. Architecture Characteristics,
3. Architecture Decisions, and
4. Design Principles.

Architectural style trivially describes which type or mix of types of software architecture patterns (SAP) are employed. Classifying a SA from this holistic point of view can be useful for categorization purposes. A traditional SAP is the *layered architecture* and

another recent one is the *microservices architecture*. Although SAPs provide the framing necessary for facilitating application-specific or business goals such as high scalability to specify the architecture of a software system in detail, its characteristics, decisions, and design principles also need to be known (Richards & Ford, 2020).

According to Richards and Ford (2020), design principles represent guidelines for development to achieve certain QRs, granting more flexible solutions for implementers as opposed to a rule set. For instance, to increase performance within a microservice architecture, a design principle may opt for asynchronous messaging between services, allowing different protocols depending on the use case. Architecture decisions instead, cover strict rules, for example permitting or denying access from an upper layer to a lower one in a layered architecture. An architecture decision generally describes the constraints of a system to avoid violations during development. Whether an exception is appropriate or not is a trade-off, as the First Law of Software Architecture states:

> Everything in software architecture is a trade-off (Richards & Ford, 2020).

Lastly, architecture characteristics correspond to QRs as defined in ISO standard 9126 and discussed by Glinz (2008). They contribute to the success of a system in a nonfunctional manner but are essential for the proper function of software. The aforementioned ISO standard describes six categories grouping a total of 27 *internal and external qualities* as listed in Table 1, which are meant to deliver optimum value. Additionally, their criticality in relation to stakeholder relevance and their impact in the event of noncompliance are evaluated for prioritization (Glinz, 2008).

Table 1: Internal and external quality, as discussed by Glinz (2008)

Functionality	**Reliability**	**Usability**	**Efficiency**	**Maintainability**	**Portability**
Suitability	Maturity	Understandability	Time behavior	Analyzability	Adaptability
Accuracy	Fault tolerance	Learnability	Resource utilization	Changeability	Installability
Interoperability	Recoverability	Operability	Efficiency compliance	Stability	Coexistence
Security	Reliability compliance	Attractiveness		Testability	Replaceability
Functionality compliance		Usability compliance		Maintainability compliance	Portability compliance

Richards and Ford (2020), however, from a practitioner's perspective, categorize those characteristics differently and dislike the term “nonfunctional requirements” due to its potential disregard by developers who are mainly responsible for the functions of a software. However, before going into the details of that topic in the architecture characteristics subchapter, the foundational concepts of SA are described in the following:

2.2.1 Foundations

The situation of ambiguous terms proceeds when refining structural the elements of SA as the literature bases itself on theoretical concepts or relates to specifications used in practice. In object-oriented programming languages such as C++ the grouping of source code is achieved by using *namespaces*, directories, and two different file types:

- Header files (`*.h`, `*.hpp`, `*.hxx`) for declaration and
- Implementation files (`*.cpp`) for application logic.

Implementation files reference header files by using the keyword `#include`. (Since standard C++20 a language feature named `module`[2] has been introduced as an alternative to header files.) To designate an organization of software parts for building a more complex structure, the term *module* is generally used to describe the grouping of code that is related code. Other authors refer to a module as a *building block*, *component*, or *subsystem*, and different programming languages employ technical terms such as *class*, *interface*, *method*, *packet*, *package*, and namespace, as already mentioned (Lilienthal, 2019). To make a strong connection to the principle of *modularity*, the author of this thesis uses “module” as a collective concept, if not distinguished elsewhere. According to Richards and Ford (2020), the SA characteristic of modularity is primarily measurable by the level of *coupling* and *cohesion*.

Coupling

As discussed before, modules group code that has a mutual relationship. Modules themselves are also related to each other, forming larger parts of a system. Their dependencies, as a result from binding intensity, define coupling. An objective of a sustainable SA is to ensure *loose coupling*, meaning fewer dependencies among the modules. This

2 https://en.cppreference.com/w/cpp/language/modules

reduces complexity and increases maintainability by superseding the need to inspect many other modules during code review and modification (Koschel et al., 2019).

Richards and Ford (2020) follow the argument of Edward Yourdon and Larry Constantine and distinguish coupling by whether dependencies are *afferent* (incoming) or *efferent* (outgoing). To illustrate this relationship, both types are drawn in Figure 2 according to Richards (2018). The first diagram shows a high level of afferent coupling in module A. That means that modules B, C and D are dependent on module A. If the inner workings of module A are changed, the other modules will probably be unable to use it without being adapted. The opposite situation occurs with module E, which is subject to changes in modules F, G or H.

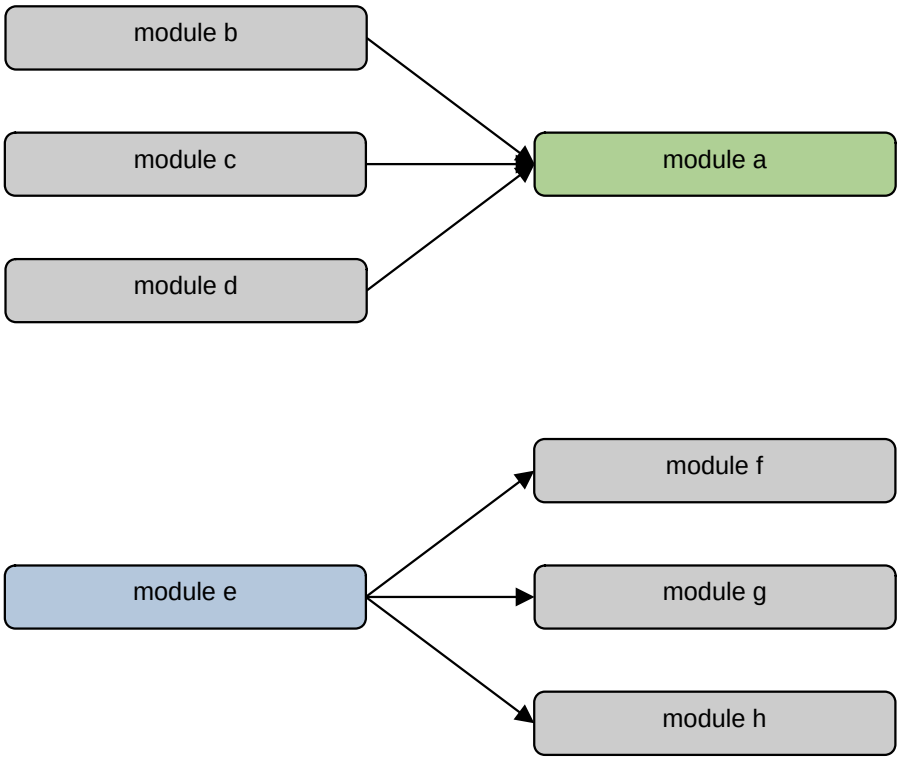

Figure 2: Afferent and efferent coupling of modules (Richards, 2018b)

Regarding higher-level architectural subsystems and design principles elaborated later, Martin (2017) relates to the type of coupling in terms of *stability*. Referring to the situation depicted in Figure 2, the upper part shows a *stable* module a which means that it is not supposed to change as it is a dependency for three other modules (A, B, and C) but

does not depend on another module itself. In contrast, module E below is likely to undergo frequent adaptation as it is influenced threefold by changes in module F to H; thus, it is considered an *unstable* module. Though one would tend to stabilize a software system as much as possible, this strategy is not practical. The objective is to construct a SA with changeable (unstable) elements on top of stable ones. Dependencies should lead to the unchangeable parts of a system or be phrased as a principle:

> Depend in the direction of stability. (Martin, 2017)

By practically taking the characteristics of the afferent and efferent coupling into account, the *Instability* (*I*) of such a structure of modules equals efferent coupling (C_e) divided by efferent and afferent coupling ($C_e + C_a$), written as an equation as follows:

$$I = \frac{C_e}{C_e + C_a}, \quad with \quad C_a: \textit{Afferent coupling}, \quad C_e: \textit{Efferent coupling}, \quad I: \textit{Instability}.$$

Thus, the instability value for the situation in Figure 2 would be low for the upper part and high for the lower part due to unbalanced ratios of efferent (outgoing) and afferent (incoming) coupling or dependencies between the modules in both cases. Those dependencies are typically represented by `#include` statements in C++ and easily countable if each source-code file contains only one class (Richards & Ford, 2020).

Cohesion

Referring to Koschel et al. (2019), loose coupling leads to highly *cohesive* modules. The Latin word *cohaerere* means “to be related” and is the origin of the term “cohesion.” Functions within a module should relate to one another but avoid calling into another module. An example is a C++ class that solves one calculation by employing several interdependent methods. It has a single responsibility, and its inner workings are independent of other modules.

Cohesion, as referred to by the Chidamber and Kemerer Lack of Cohesion in Methods (LCOM) metric, is the “sum of sets of methods not shared via sharing fields” (Richards & Ford, 2020) and is illustrated using an example with octagons for fields and squares for methods in Figure 3. In that figure, class X is an ideal situation with a low LCOM score and, thus, high structural cohesion. Fields are intensely shared among methods

and form a coherent class. Class Y, instead, lacks cohesion, making each field and method pair a candidate for refactoring. The reason is that each pair could be packaged separately in its own class without affecting the application's behavior. The same applies to the last pair in Class Z, while the rest are coherent. Class Z depicts a situation of mixed cohesion (Richards & Ford, 2020).

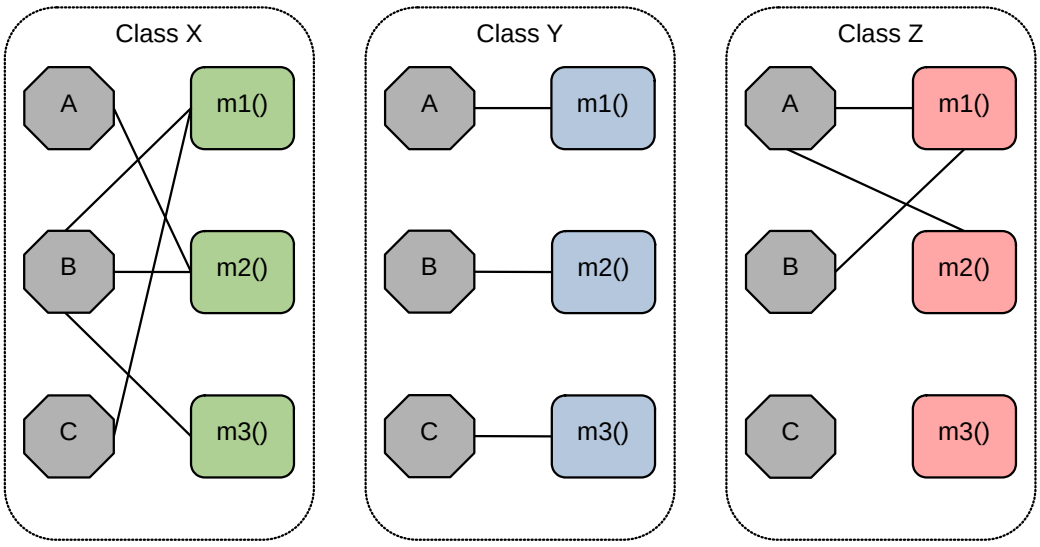

Figure 3: LCOM metric example according to Richards and Ford (2020)

Although there have been many approaches to finding distinct measurement methods in computer science, cohesion, on the other hand, is still a less precise metric. The LCOM metric mentioned above can support moving from one architectural style to another by identifying classes that are coupled incidentally. But in general, software metrics such as LCOM only work on a structural basis and cannot determine a logical lack of cohesion. Richards and Ford (2020) refer to that fact in their Second Law of Software Architecture: “Why is more important than how.” Employing the LCOM metric to find low or missing cohesion does not explain possible rationales for such a decision.

Practically speaking, Lilienthal (2019) hints at the fact that the module name already indicates the state of cohesion. If a module (a class in a C++ project) provides a name that describes its task appropriately, it can be assumed that a software system has been designed considering modularity, and vice versa. Typical bad names are “Manager,” “Handler,” “Util,” or “Helper.” All of these share the concept of being unable to find their

particular responsibility and end up being repositories for code relics. The opposite happens with duplicated code. If the same task is performed by more than one module, there is no single responsibility, as the same code is shared by different actors. Code relics and duplicated code often occur at an early stage of development and are not cleaned up later. Clean code principles can remind developers to watch out for and address of such deteriorating code structures. While working collaboratively in the codebase, architects are supposed to support the micro-level cleanliness of developers by designing and monitoring maintainable connections between those code structures. Therefore, as already indicated by referring to Martin's definition of stable and unstable modules in the section regarding coupling, modularity and its measurement also apply to higher-level architectural elements known as *components*, which are described next.

Components

Lilienthal (2019) refers to a *component* as subsystem that is characterized by a unique name and public interface for interaction within the code base, as defined by the International Software Architecture Qualification Board. However, from the viewpoint of a pragmatic architect, component broadly refers to the physical manifestation of a module. In C++ programming, this might be a *library*, as it bundles classes at a higher layer of modularity. Furthermore, a TCP/IP service deployed as standalone units, can appear as a component in a SA. There are no rules mandating the use of components, but an architect should consider these as an additional layer of modularity for partitioning a complex system (Richards, 2018a).

Components support decomposing a system from the top-level to create either technical- or domain-specific partitions. A system might be partitioned into technical capabilities (presentation, business rules, services, persistence, etc.) to organize the code into layers for separation of technical concerns. This style is called layered architecture and matches the model view controller[3] design pattern. Figure 4 contrasts it with another way of partitioning, which is inspired by domain-driven design[4] (DDD) and strives to identify components at the domain level. The rationale is to model the problem domain and its relations by creating a matching software structure (Evans, 2003).

3 https://martinfowler.com/eaaDev/uiArchs.html
4 https://www.dddcommunity.org/learning-ddd/what_is_ddd/

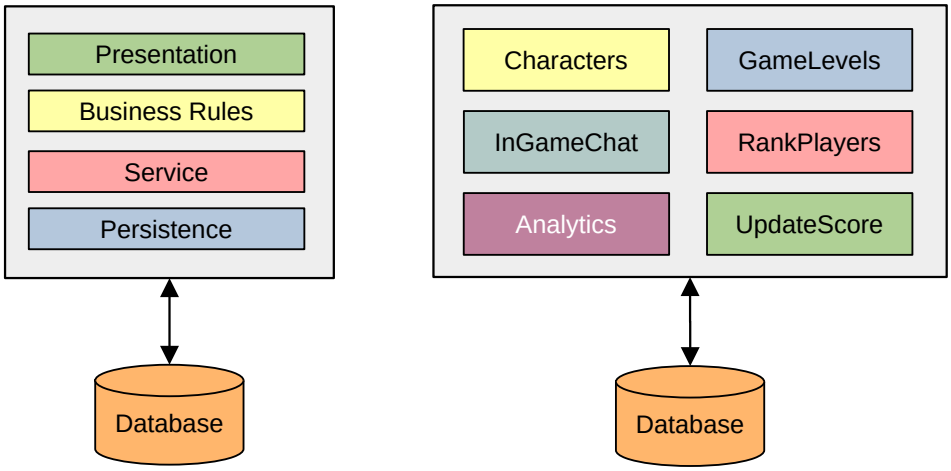

Figure 4: Components in a layered architecture and in a Domain-Driven Design

Consider a role-playing online game, as depicted on right-hand in Figure 4. One domain would be the game levels. Components in DDD represent distinct domains or workflows to achieve a more realistic and layer-independent abstraction. A workflow, such as dynamically generating a game level, often takes place at more than one technical layer. Partitioning a system that way can minimize code scattering, as components comprise any code that represents domains instead of grouping technical aspects. DDD seems to have been adopted in industry lately. However, depending on the organization's characteristics, a layered architecture can still be reasonable (Richards & Ford, 2020).

2.2.2 Architectural Styles

Also known as software architecture patterns (SAPs), architectural styles try to subsume SA aspects by defining a descriptive term to serve as shorthand between architects. This classification can communicate implicit details, including beneficial and adverse architectural characteristics. Additionally, a pattern is useful for easier recognition of a software structure in practice. Visualizing dependencies can help create an overview of the SA and hint at intentional boundaries in the system design. Conversely, the absence of any traceable structure is an *antipattern* (a pattern that yields negative results on the long term), which is classified as “big ball of mud” and commonly encountered in projects that began as scripting solutions. Software can unnoticeably tend to this situation during its evolution. The visualization shown in Figure 5 is an extreme example of

the big ball of mud anti-pattern. This architecture lacks any structural characteristic and for that reason does not support maintenance and sustainability in long term. The classes have cyclic dependencies shown as lines, which are outside specific components or domains indicated by the four colors yellow, red, green, and blue. Changes to such a system are errorprone (Richards & Ford, 2020).

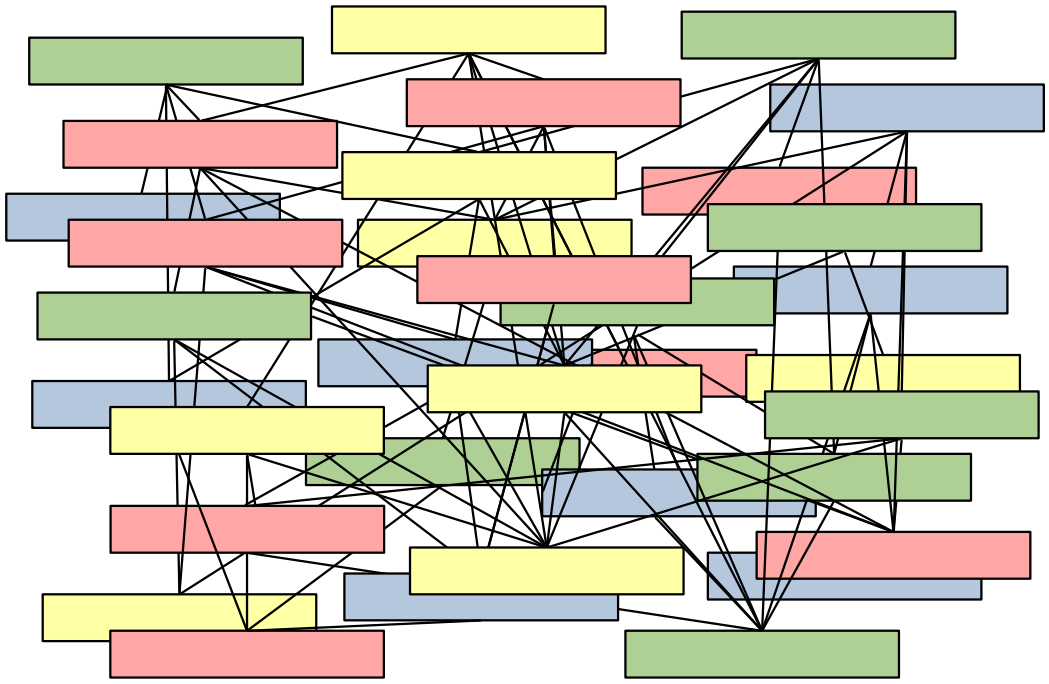

Figure 5: Example big ball of mud architecture

Another objective of patterns is to provide a reusable solution base for typical problems while avoiding the violations of the aforementioned SA principles. Many architectural styles, or SAPs, comprise fundamental patterns or build on them. Some of them are related to historical milestones in computer applications. For example, personal computer software began as unitary architecture and changed to a client/server structure with the advent of networking and distributed systems. That two-tier architecture allowed front-end and back-end separation into desktop client and database server or web browser and web server. The latter has evolved into a three-tier system using tools like JavaScript with an application server separate from a database server to, for example, account for scalability and security requirements (Richards & Ford, 2020).

The layered architecture, as depicted on the left in Figure 4, is a traditional style for creating a hierarchical structure. From top to bottom, the typical layers are presentation, business rules, service, and persistence. Each layer can only access the next lower layer. Usually, an operating system (OS) is built that way to provide abstraction according to the ISO open systems interconnection model[5]. However, application-level software uses interlayer skipping for performance reasons and tends to use mixed layering (horizontal and vertical) for larger systems (Lilienthal, 2019).

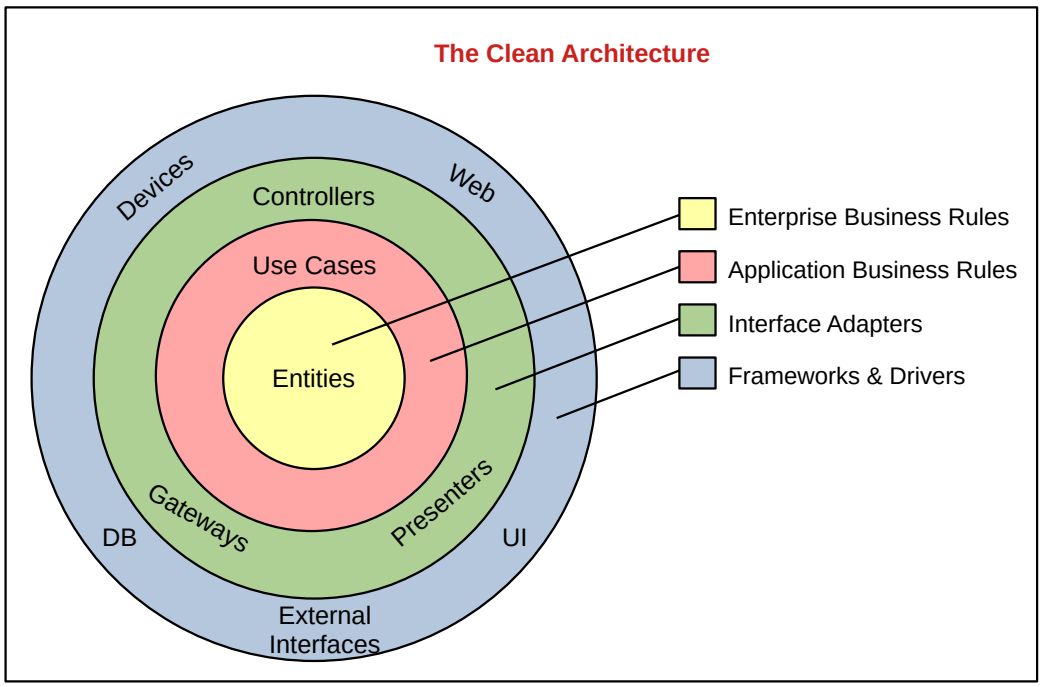

Figure 6: The Clean Architecture (Martin, 2017)

The clean architecture of Martin (2017) is another approach to enhancing the layered architecture towards fewer dependencies from a domain point of view. Modularity is mainly achieved by separating the business logic from the technical details. With enterprise business rules as most general high-level entities in the center of concentric circles shown in Figure 6, clean architecture defines a dependency rule as follows:

> Source code dependencies must point only inward, toward higher-level policies. (Martin, 2017)

5 https://osi-model.com

This means that functions, classes, and variables of outer circles cannot be used by an inner circle to prevent critical inner business related policies from breaking, for example in the event of changing technology. For example, frameworks providing a user interface (presentation layer) or a specific database (persistence layer) should be easily replaceable without affecting the next inner circle of interface adapters. Clean architecture, as an architectural style, aims to provide business continuity and can be supported by architecture characteristics, as described in the following subchapter (Martin, 2017).

2.2.3 Architecture Characteristics

Software projects commonly begin with requirements engineering[6] to identify factors of the system to efficiently address the problem domain while considering business needs. However, those aspects do not necessarily reflect concerns that are important to create a successful SA. An architecture characteristic takes design decisions apart from the domain into consideration, has a certain impact on the structure, and is critical holistically for the success of a system. Specifically, architecture characteristics range from lower structural requirements at the code level (modularity) to higher concerns of infrastructure and operations (scalability). The categorization by Richards and Ford (2020) is slightly different to the ISO standard 9126 described in Table 1 to match the perspective of an architect. Those in addition to the standard are listed and outlined in the following.

Operational Architecture Characteristics

- *Availability*: How long and uninterruptedly can the system be used? (e.g. 24/7)
- *Continuity*: Can the system still be used in case of a disaster?
- *Performance*: What load capacity is a system supposed to provide?
- *Reliability/Safety*: How critical is the system operation for business continuity?
- *Robustness*: Can the system handle, for example, hardware failures or power outages?
- *Scalability*: Is the system's performance and operation guaranteed, if users or requests increase?

6 https://dl.acm.org/doi/10.1145/336512.336523

Structural Architecture Characteristics

- *Configurability*: How easily can end users change the system's configuration?
- *Extensibility*: To what extent is the addition of new features important?
- *Leveragability/Reuse*: Can parts of the system be reused with other products?
- *Localization*: Does the system support multiple languages with different character sets and units of measure?
- *Upgradeability*: What duration or effort must be used to upgrade the system?

Cross-Cutting Architecture Characteristics

- *Accessibility*: Can the system be used by disabled users (e.g., colorblind or hearing-impaired people)?
- *Archivability*: What is the strategy for data retention?
- *Legal*: Are there regulations to comply with (e.g., data protection)?
- *Privacy*: What level of data protection between system users and between users and operators can be guaranteed?
- *Supportability*: How much technical support is needed, and to what extent is logging etc., involved in debugging system errors?

Richards and Ford (2020) distinguish an architecture characteristic by whether it is implicit or explicit. For example, professional audio live recording is absolutely dependent on low latency along the whole signal chain, although a software project in that field of application would not explicitly specify low latency requirements for all its systems, as this requirement is implicit. In contrast, that project might list high security for strategically positioning itself on the market without considering some contradicting aspects. Security improvements may have a negative impact on performance, leading to high latencies since encryption and the hiding of secrets demands processing capacity. Regarding the pro-audio project, that probably induces restraint from high security requirements in the trade-off process. An architect needs to be aware of critical requirements and choose the fewest architecture characteristics possible to create a capable but less complex SA.

Whenever architecture characteristics are gathered they must first be dissected into smaller ones if composite and then evaluated and adapted to the specific needs of a project since every one of them is subject to interpretation depending on perspective. Thus, overlapping definitions and a changing list of requirements introduced here as architecture characteristics are issues to manage as an architect. DDD can support dealing with the ambiguity of terms by leveraging the ubiquitous language for communication (Richards & Ford, 2020).

2.2.4 Architecture Decisions

As explained before, the choices concerning construction techniques, dependencies, interfaces, structure, technical, and nonfunctional characteristics significantly affect a software system and are considered as architecture decisions. For example, construction techniques are platforms, frameworks, tools and even process models that impact the system in a particular manner. Architecture decisions guide development teams whenever technical choices appear during implementation. Thus, architects are responsible for effectively communicating, plausible justifying and transparently documenting decisions. Wiki page templates and short files in plain text, AsciiDoc[7], or Markdown[8] format can be used to collect an *architecture decision record* (ADR)[9]. ADRs were introduced by Michael Nygard in 2011 and use the following sections (Richards & Ford, 2020):

- *Title*: A short and precise description of a sequentially numbered decision
- *Status*: For example, proposed, accepted, discarded, deprecated, or superseded
- *Context*: Conditions and reasons leading to that decision
- *Decision*: The result of the previously mentioned contextual considerations
- *Consequences*: Positive and negative implications of the decision

A new ADR overrules its predecessor, and tooling can help connect these ADRs to creating a history of decisions documenting the project's evolution. Consider a simple project for the bulk renaming of files and directories on a Unix-like system that began as Bash script. Using it with several operating system distributions has shown that some

7 https://asciidoc.org
8 https://commonmark.org
9 https://adr.github.io

commands are not available in different shells. Therefore, the developers decided to use Python instead. Assuming that the usage of Bash has been previously documented as ADR 5, the new one could be titled “ADR 6: Use Python language” and have the status “accepted, supersedes 5.” ADR 5 becomes obsolete, and its status would be adapted to “superseded.” The mentioned rationales become the context; a consequence might be interoperability, and developers need to learn Python (Richards & Ford, 2020).

Though extendable by more sections, an ADR providing title, status, context, decision, and consequences is usually sufficient for communicating who has decided what, when, and why. Ideally, stakeholders can answer those questions by solely reading ADRs and understanding their justifications. With that said, affirmative formulation and a comprehensible but short discussion of pros and cons are preferred. In connection with a Wiki or version control system, supplemental meta-data (date, author, etc.) can help understand the scope and history of ADRs (Richards & Ford, 2020). In addition to this way of conveying rationale, developers are guided by design principles during implementation.

2.2.5 Design Principles

Writing “clean code,” as strongly advocated by Martin (2017), is important for a sustainable software system on a micro level. However, it cannot account for the interaction of higher-level structural elements. The SOLID principles are a collection of design principles for the clean architecture. The acronym stands for the following principles:

- **S**ingle responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

Each principle supports distinct architectural considerations by representing guidelines for arranging functions and data structures and choosing appropriate interconnections. Generally, the SOLID principles should be taken as advice rather than strict rules as environmental factors can shape their adherence (Martin, 2017).

2.2.6 Environment

Another interdependent impact on a software system as defined by ISO (2011) is the "(system) context determining the setting and circumstances of all influences upon a system," which is summarized by the term *environment* and includes factors regarding the development organization and application operation. Likewise, several non-technical aspects, such as office politics, economic, and social influences, affect a system.

Infrastructure

Most software projects operate on a vast amount of data and thus, rely on separately managed data storage such as databases. Consequentially, data structures and data architectures are related to SA and need to be designed in a complementary manner. With that said, there is an interconnection between architecting software code and operating its data storage systems for the whole lifecycle (Richards & Ford, 2020).

Additionally, a software system is typically embedded in a landscape of evolving applications and other changing projects. If one of them is initiated or terminated, chances are that a SA may completely fail without adaptation to such situations. Most of the time, large software is not just an isolated system but a system of systems. Or a software constitutes a recursive construction: It is built upon other software that is relying on major software components made of smaller software modules, etc. (Martin, 2017).

Technically, considering the environment of a SA also means dealing with platforms for operation and the runtime aspects themselves. Some projects are designed to run on existing hardware, while others require new platforms. The same applies to the chosen development environments and tooling for implementation, building, and testing. Depending on the project, those tools must also be adapted or extended in order to meet architectural characteristics for proper operation (Koschel et al., 2019).

Project Management

Another constraint is connected to budgets. In the words of Brian Foote and Joseph Yoder, as cited by Martin (2017):

> If you think good architecture is expensive, try bad architecture.

The costs of software development are not easy to estimate. There are limited resources to realize a project within a given amount of time and effort. Budgets can define those limits for good and bad reasons. A good reason would be to prevent developers from implementing more functions than necessary, whereas a bad reason would be to cut costs by treating a software system as a disposable product. The latter would probably lead to software that is difficult to change and, thus, unmaintainable. Both examples would be subject to a SA design bound by budget constraints becoming a noticeable issue during implementation and operation (Martin, 2017).

Speaking of software as a product, its project management affects SA by strategically controlling which architecture characteristics are of high priority and which are rather negligible. As these priorities are dynamic, an architecture should gracefully handle changes during the development project and interact with them to identify conflicting requirements (Koschel et al., 2019).

Office Politics

Influences such as social structures in an office may also reflect on SA. Particularly seasoned software projects often keep their “parents,” developers who made significant decisions throughout their evolution. These stakeholders tend to protect their “baby” and might be an obstacle concerning new ideas. However, their knowledge about a system's inner workings is important in the course of change (Lilienthal, 2019).

Additionally, the overconfidence of modern developers is a threat to a sustainable SA, as Martin (2017) observes. The self-deceiving assumption that messy code can be cleaned up later allows faster time to market but decreases the performance of development in the long term. Developers are supposed to write maintainable code as opposed to finding workarounds until a system has to eventually be redesigned from scratch. It is their responsibility as stakeholders to provide and preserve clean code to provide understanding to other developers interacting with it.

2.2.7 Process

Lastly, the process of software development itself can influence SA and is dependent on it as well. For instance, the agile[10], continuous integration (CI)[11] or waterfall[12] methodology can affect the SA, and vice versa. Richards and Ford (2020) state that many projects have recently adopted agile development to meet software's nature and gain faster feedback. Iterative processes can deal with recurring changes and better use the input from feedback, especially for architecture decisions. Though projects employing monolithic architectures tend to stick with traditional methodologies, agile processes come into action when migrating to modern architectures for quality and productivity reasons, according to Martin (2017). To this end, *Test-Driven Development* (TDD) can assist in quality control from the beginning of implementation as acceptance tests ensure the intended functionality, and other test types can catch exceptions. Jason Gorman showed that using TDD also improves efficiency in the long term.

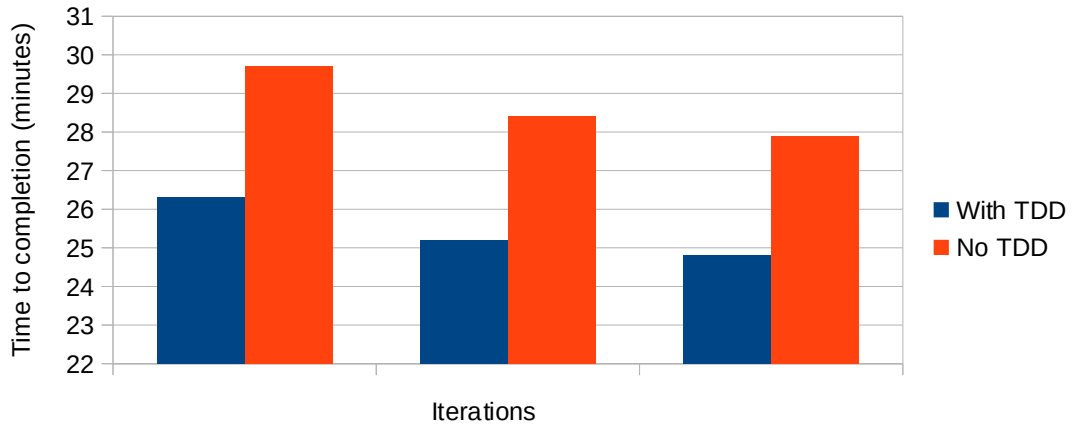

Figure 7: The impact of using TDD towards time to completion (Martin, 2017)

Gorman wrote a simple program for converting integers into Roman numerals over a period of six days and measured his completion time with and without TDD. A predefined set of acceptance tests served as a completion indicator. As illustrated in Figure 7, even the slowest TDD day was faster than the fastest non-TDD day, and on the average, he proceeded approximately 10% faster with TDD than without it (Martin, 2017).

10 https://agilemanifesto.org
11 https://martinfowler.com/articles/continuousIntegration.html
12 https://dl.acm.org/doi/10.5555/41765.41801

2.2.8 The Role of an Architect

The same amount of imprecision for defining SA itself, as introduced in the beginning of this subchapter, also applies to the software architect's role due to the fact that software engineering, and especially *software architecting*, are described as a:

> process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system's life cycle. (ISO, 2011)

These are young disciplines. For that reason there is a lack of long-term knowledge in comparison to, for example, construction. Software architecting, however, can be related to typical tasks of construction if considering the SA description as the blueprint of a system. By the very nature of an iterative process, that blueprint needs to be adapted to the actual SA and vice versa, employing a software architect with continuous communication, discussion, and documentation of the design. The architect's holistic view on a system allow him or her to support the project manager in planning and assigning work while considering risks and their mitigation (Koschel et al., 2019).

Further, an architect is typically expected to act at the intersection of development and business strategy, which shifts with every project within the same company and within different companies. However, some important core skills that form a software architect's profile provide guidance for choosing technology according to architectural characteristics, preservation of the vitality of an architecture by frequent analysis and adaptation, and verifying adherence to documented architecture decisions and design principles. Additionally, an architect must be familiar with business concerns for effective communication and be capable of conveying ideas, decisions, and principles while understanding office politics on the one hand. On the other, he or she must lead, coach and mentor developers during implementation while being up to date with the latest technology trends and changes in the problem domain (Richards & Ford, 2020).

Lilienthal (2019) points out that architects and developers need to cooperate. The latter can benefit from architects' experience as developers. Conversely, architects are supposed to participate in programming if possible and be available for consultation. The notion behind this is to collect information regarding changes to the planned architecture proactively. In one word, architecting could be characterized as *Holschuld* (German

for “a collectable debt”). Ideally, retrospectives are part of every iteration, and team sessions for discussion and training are organized to take place regularly. In discussions, architects can communicate the state of their architecture while constantly analyzing its evolution and measuring trends.

2.3 Analysis and Recovery of Software Architecture

General measurements of coupling, cohesion, complexity, and size, or their combinations, are used to derive metrics, according to a literature review by Coulin et al. (2019). *Software architecture analysis* (SAA) describes techniques, such as static code analysis, to generate metrics from available source code. These are used to find indicators for deviations in the implementation. To extract the as-is architecture of a system, s*oftware architecture recovery* (SAR) is employed. Drifts and violations of an intended SA typically arise from suboptimal decisions during architecting and development, causing issues at the code level, which are measurable by tools and named after a relation known from finance.

2.3.1 Technical Debt

Using the metaphor *technical debt* (TD) for describing decisions with long-term implications facilitates its understanding for business-people. As the interest rate for financial debt is a function of time, TD likewise accumulates if not paid back. In software development, this requires constantly analyzing and improving a system as TD cannot be avoided completely. It occurs at the code level through quick and dirty hacks of new features or at the macro level via architectural trade-offs affecting a software system's maintenance and expansion capabilities (Tornhill, 2018).

Development teams are supposed to frequently address TD while extending a software system with a focus on quality. Lilienthal (2019) states that a balance of these activities supports a software system to remain in a “corridor of low technical debt with a predictable maintenance effort” and prevents frustration of developers. Otherwise, bug fixes are difficult to achieve; functionality cannot be implemented easily; and changes are perceived as a painful struggle. If a system has reached that state, it has dramatically deviated from its intended design and is slowly eroding as compared to a building.

2.3.2 Architecture Erosion

The erosion or degradation of SA is a natural process during the lifetime of a system as complexity grows and requirements change. Whiting and Andrews (2020) describe that *architectural erosion* (AE) emerges from decisions violating the intended SA and most probably breaks the system. According to Li et al. (2022), AE manifests at several structural levels, leading to terms such as “code decay” and “modular deterioration,” and does not depend on the size of the codebase. Larger software systems may be complex and employ code from other projects, but they do not naturally entail erosion of their architecture for that reason. Conversely, there is no guarantee that small systems will be immune. AE often happens insidiously without affecting obvious characteristics, such as performance. Accepting TD in order to deliver the next release is an example of intentional decisions with long-term implications for SA. Additionally, developers sometimes accidentally code outside of architectural boundaries. Both situations can cause AE, though they do not affect the performance of a system. A system suffering from AE is difficult to understand and maintain as the principles of the intended SA are not adhered to. However, AE occurs in all phases of development and may originate as early as in the design stage or during architectural considerations, while unnoticeably evolving with subsequent versions of a software system (Li et al., 2022).

2.3.3 Architectural Smells

A suboptimal decision during software design is called *architectural smell* (AS). Several studies link architectural smells to AE, claiming that TD mainly arises from architectural issues. An AS is also known as “design smell,” “antipattern,” or, generally, a “bad smell” and differs from a code smell as it does not apply on a single code structure. Instead, higher-level modules and components are affected (Azadi et al., 2019). Le et al. (2017) classify bad smells according to the architectural concepts of modularity and coupling. In this work, the category of dependency-based smells is considered. A *dependency cycle* represents circularly linked components violating the modularity principle. If one component is changed, adaptations in the whole cycle are necessary. The same violation happens if a component has an excessive number of outgoing or incoming links to other components, which is called a *link overload* (Le et al., 2017).

2.3.4 Symptoms and Reasons for Architectural Erosion

Apart from code smells, architectural smells indicate a probability of AE. They are regarded as the second most prevalent symptom, as classified by Baabad et al. (2020), and further distinguished into three groups: architectural bad smells/architecture anti-patterns, architectural change/instability, and architectural hotspots. In this group, architectural bad smells occur in 47.06% and architectural change/instability in 41.18% of the investigated primary studies. The former has a strong relationship to one of the reasons also researched by Baabad et al. (2020). With a score of almost one fifth a rushed evolution is the primary cause of AE and is characterized by a number of architectural problems such as an increase in smells through successive system versions. Table 4 lists the five most prominent reasons related to AE in open source (Baabad et al., 2020).

Table 2: Most prominent reasons for AE in OSS according to Baabad et al. (2020)

Order	Prevalence	Reason
1	19.77%	Rushed evolution
2	18.60%	Recurring changes
3	12.79%	Lack of developers' awareness
4	9.30%	Time pressure
5	9.30%	Accumulation of design decisions

Secondly, due to recurring changes, development activities are prone to mistakes that negatively affect the architecture. These arise with new technologies or occur in the adoption of new functionalities and features, for example. Further, their addition is either irresponsible or unintended, and changes in their implementation or removal are uncontrolled or unsuitable. Furthermore, architectural design decisions may be modified. A lack of developers' awareness of AE leads to severe problems that are not investigated during the SA evolution, representing a tertiary reason. This may be related to the adoption and selection of inexperienced developers, a lack of understanding of architectural basics, or insufficient business knowledge. The generation of OSS as a distraction or hobby may be practiced and trained, and writing code that is either unsuitable or improper is a potential source for explanation. Lastly, developers may not practice long-term commitment to the project. Another significant reason for AE, according to Baabad et al. (2020), is time pressure. Deadlines, time constraints, and workloads

may result in temporary solutions in the codebase that contribute to an increasing number of architectural debts impacting SA over time. To the same extent, the consciously or unconsciously acquired accumulation of design decisions deteriorates quality attributes such as evolvability and maintainability. Other reasons have a less than 5% prevalence and are not considered in this work (Baabad et al., 2020).

With special regard to academic software Laser et al. (2021) observe how other forces shape OSS development in research projects. As already introduced in 2.1.3 Academic Projects, they found the need to “publish or perish” (Laser et al., 2021) to be an explanation for a lack of emphasize on the adoption of practices geared towards understanding, controlling, and reducing AE. Further, they describe how small academic programs for proving scientific concepts can become complex software systems referred to as large, long-lived academic research tool suites (3L-ARTS). This process may begin when an existing tool is picked up by a PhD student and used as a research foundation. Different research objectives urge the need to extend and change the tool in a way that might not comply with original design decisions, and “independently developed extensions of a core pool of capabilities result in progressively increasing departures from the original architecture” (Laser et al., 2021) leading to a *software archipelago*.

Archipelago model

Laser et al. describe an archipelago as a set of software tools that address closely related problems within a research topic. Each island (software tool) represents an outcrop of another one and is thus, highly coupled to it. Adding new islands induces more direct and indirect coupling between islands in the set. This draws the following consequence:

> [A] person attempting to use one of these islands typically needs to understand and comply with a large subset, if not all, of the archipelago's architecture. (Laser et al., 2021)

However, different people have developed each island as a solution for different purposes and used it in different ways, which leads to inconsistent or conflicting architectural decisions in the archipelago. For that reason, students may decide to build new systems from scratch rather than to refactor and reuse the archipelago. Laser et al. provide a categorization by reviewing several 3L-ARTS from the literature.

Figure 8 depicts an example archipelago created using Azgaar's Fantasy Map Generator[13] for illustration purposes. *O* indicates the original project in the set of islands; it may have been produced by one or more PhD students and described in a number of publications. Major or minor extensions due to additional publications are designated by *OM* or *Om*, respectively. Major extensions typically yield one or more peer-reviewed publications or an entire dissertation as they are introduced by new contributors, such as students or post-docs, after initial developers have left the project. In order to provide a solution for a problem that has previously been unsolved, new functionality is implemented with significant additions by either leveraging O's existing APIs or by adapting them. If it is directly enabled by O, it is regarded as type OM_1. Instead, type OM_2 is an extension that requires adaptions to the APIs (Laser et al., 2021).

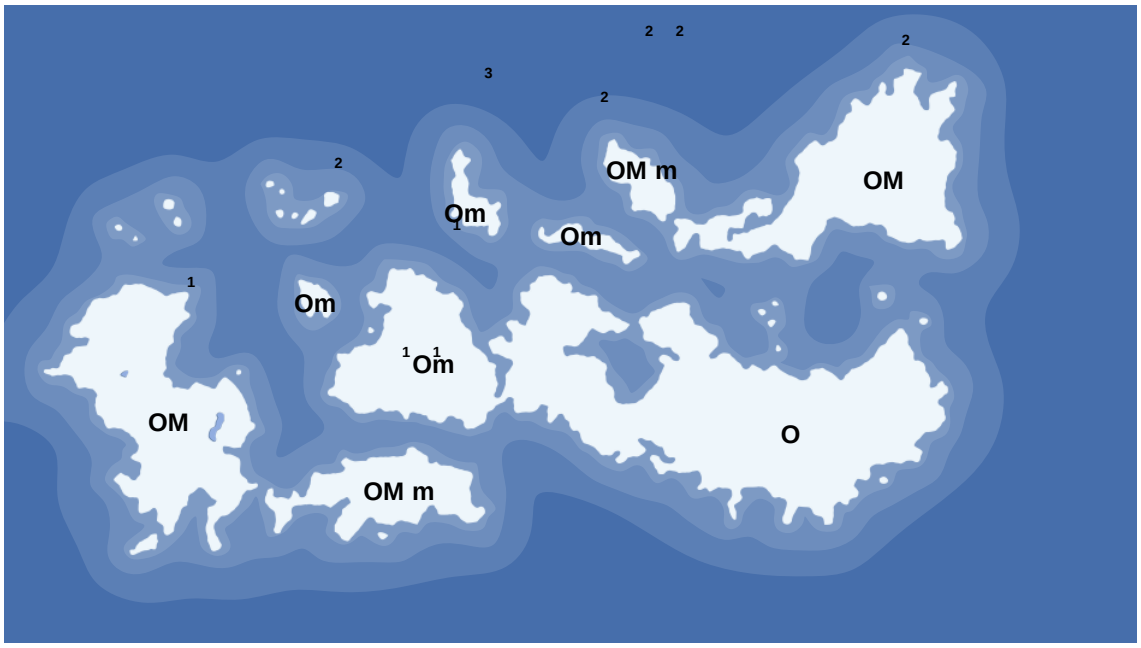

Figure 8: Archipegalo model of 3L-ARTS according to Laser et al. (2021)

The original authors or new contributors may also introduce minor extensions that represent less than moderate deviations from O and typically publish no more than one paper attempting to generalize or improve the results of prior work. Similar to major extensions, minor ones are distinguished by their deviation from the original project. Minor extensions leveraging its APIs are of type Om_1, and those with slight adaptation or addition of relatively simple functionality are regarded as type Om_2. Type Om_3 is char-

13 https://azgaar.github.io/Fantasy-Map-Generator

acterized by modifications for supporting additional inputs or outputs or combinations with third-party utilities. Though this type is inspired by O, it deviates significantly. A combination of types, such as a minor extension to a major extension, is depicted in Figure 8, and a major extension to a major extension of O may also be possible, although it is not illustrated.

An archipelago of software tools promises quick extensions by new contributors using their own choice of tools and technologies as only existing APIs evince architectural constraints. However, a consequence is that different technologies may be used without considering the complexity of later code management, and the design decisions of previous contributors may be neglected or even violated, inducing AE (Laser et al., 2021).

2.3.5 Architecture Recovery Techniques

Once the architecture of a software system is lost, analysis can become cumbersome. Software architecture recovery (SAR) can extract the architectural structure from source code files to apply metrics and verify whether a desired architectural style is respected. Using SAR in circulation with software architecture analysis (SAA) and TD removal on a regular basis, as illustrated in Figure 9, is an option to integrate countermeasures for improving the architecture during release cycles. However, the informative value of a recovered SA from as-is dependencies in source code is subject to the used clustering algorithms (Link et al., 2019).

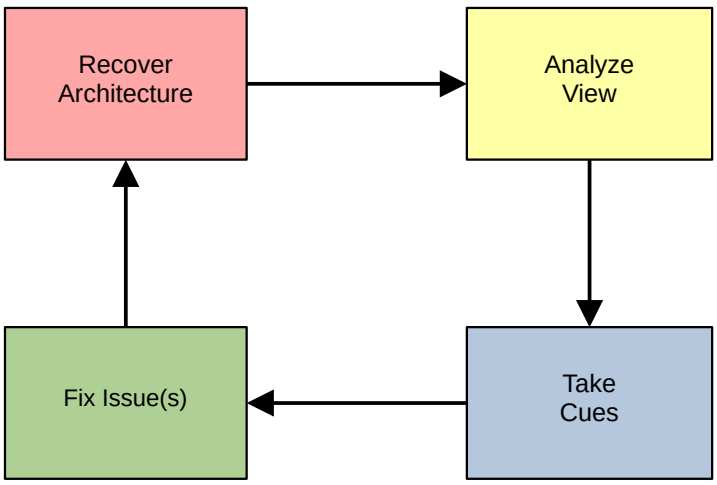

Figure 9: Circulation of SAR, SAA and TD removal (Link et al., 2019)

Algorithm for Comprehension-Driven Clustering

In a paper from 2000, Tzerpos and Holt describe an *algorithm for comprehension-driven clustering* (ACDC) based on structural patterns derived from common observations in manual decomposition of software systems. Their structure-based approach considers procedures or variables as nodes and relations between them as edges of a vertex of a graph to create clusters by partitioning it into subgraphs. A cluster is built from a dominant node identified by a high number of edges to other nodes, which are called *subgraph dominators* (dominator nodes and their dominated nodes). For the naming of a cluster, the algorithm leverages filenames within that partition to allow better recognition by developers who are familiar with the codebase. If a cluster exceeds the limit of 20 files, it is partitioned further in order to retain a manageable size for comprehension. Beyond that, ACDC considers the following structural aspects for clustering:

- Code entities found within the same source file
- Source files in the same directory
- Body files with associated header files
- Files serving a similar purpose (e.g., drivers for various peripherals)
- Support libraries accessed by a majority of other components
- Entities depending on a large number of other resources, such as a driver

Tzerpos and Holt (2000) mention that some of these patterns must be used with care and may not emerge in small systems as the testing included large systems such as Linux 2.0.27a with 955 source files and 750,000 lines of code.

Package Structure

The *package structure* (PKG) algorithm can also represent the implementation view of the as-is SA by using the organization of source files in directories for the recovery. Each directory is regarded as a component, and references from source files within that directory to source files in another directory represent dependencies. Compared to the former, it is a simple technique that does not account for conceptional views. However, the directory structure of implementation files can serve as a baseline for assessing the results of ACDC (Le et al., 2015).

2.3.6 Architectural Metrics

Recovering the implemented SA for a range of software versions allows further investigation of architectural change at different levels of structural abstraction. The addition, removal, and modification of components is regarded as system-level architectural change and measured by the *architecture-to-architecture* (a2a) metric (Le et al., 2015).

System-level Metrics

In the following equation, the a2a metric calculates the distance from one architecture to another by detecting the minimum transform operation (*mto*) number and using a percentage value from 0 for none to 100 for absolute similarity (Link et al., 2019):

$$a2a(A_1, A_2) = \left(1 - \frac{mto(A_1, A_2)}{mto(A_\varnothing, A_1) + mto(A_\varnothing, A_2)}\right) \times 100\% \quad with$$

$$mto(A_1, A_2) = remC(A_1, A_2) + addC(A_1, A_2) + remE(A_1, A_2) + addE(A_1, A_2) + movE(A_1, A_2).$$

The mto value for transforming architecture A_1 into A_2 is the sum of five operations regarding implementation-level entities (indicated by a capital *E*) and clusters (*C*):

1. additions (*addE*),
2. removals (*remE*),
3. moves (*movE*),
4. additions (*addC*), and
5. removals (*remC*),

where each addition and removal of an implementation-level entity comprises two subsequent operations. In addition, the process is to

1.1. add an entity to the architecture and

1.2. move it into the cluster.

During the removal, the process is to

2.1. move an entity out of its current cluster and

2.2. remove it from the architecture.

For normalization, the result of *mto(A_1, A_2)* in the a2a equation is divided by the sum of each architecture's individual distance A_i to a null architecture $A_\varnothing$ by determining the number of required operations for that transformation. For example, *mto($A_\varnothing$, A_1)* and *mto($A_\varnothing$, A_2)* specify the distance of architecture A_1 or A_2, respectively, from $A_\varnothing$ as the point of reference. Distance measurement between two architectures allows comparison from a holistic point of view. To assess more details in a distinct architecture, measurements can be applied to the hierarchically next lower level (Link et al., 2019).

Component-level Metrics

The *cluster-to-cluster* (c2c) metric is used to assess the amount of similarity between implementation files within two components (or clusters). It is used in the calculation of the *cluster coverage* (cvg) value, which represents the overlap of one architecture's cluster with that of another one and is determined as a percentage value. Practically speaking, cvg can reveal the existence of a component in an earlier software version or its introduction in a later one (Link et al., 2019). A strong predictor for the change of a component is the instability metric, as introduced in the coupling section of subchapter 2.2.1. A high level of instability makes AE likely. This is due to the fact that frequent adaptation is necessary if a component has many dependencies and is forced to change in the case of modifications to them (Le et al., 2016).

Another relational metric is the amount of coupling and cohesion between and within components, which is known as *modularization quality* (MQ) and is used in the Bunch recovery technique by Brian Mitchell. *BasicMQ* is a calculation variant for which *intraconnectivity* describes cohesion within components while *interconnectivity* (coupling) is regarded as cohesion between components. For BasicMQ the intraconnectivity of components is related to their interconnectivity and measured as numerical values between −1 and 1. As long as a positive value is obtained, the cohesion is higher than the coupling. Negative values mean the opposite. *TurboMQ*, instead, is a ratio for intraconnectivity divided by interconnectivity and varies from 0 to 1. Thus, higher values indicate a higher amount of intraconnectivity over interconnectivity and reflect cohesive components that are less coupled (Mitchell & Mancoridis, 2006).

3 On the Methodology of This Thesis

The theoretical background reveals that the phenomenon of AE (architecture erosion) is entangled with the process of software development in a real-life setting and influenced by the environment and context of a software project. Therefore, an empirical approach with data collection from multiple sources and no control over certain variables, as opposed to an experiment, is necessary. An appropriate research method for such a situation is a single case study as defined by Yin (2018). This chapter deals with the design of the case study, its data sources and strategies for investigation. Additionally, the limitations of the chosen methods are described.

3.1 Case Selection

In software engineering, an explanatory case study can be employed to understand the occurrence of a phenomenon, for example by identifying a causal relationship. Beginning with the existing theory of AE reasons and deriving hypotheses and research questions in the introduction of this work, data collection and their interpretation are subsequent steps in this deductive approach (Runeson et al., 2012a). Selecting an appropriate case is important to relate to theory by showing extreme or unusual instances (Runeson et al., 2012d). Thus, the JackTrip project is selected for the following reasons:

- There are no constraints regarding availability as it is an actively developed OSS that publicly shares source code, version control data, and documentation.
- Initial measurements imply that symptoms of AE occurr or may occur for one or more suggested reasons, making it a contemporary example.
- The academic origin seems to be a special characteristic of the chosen project and might be related to the occurrence of AE, according to literature.

In particular, the product line of the JackTrip application from the first stable version 1.1 up to the latest version 1.6.8 (released on December 6th, 2022) is the primary object in this case study. Besides the git repository[14], other data sources provide documentation and contextual information to be analyzed as follows.

14 https://github.com/jacktrip/jacktrip

3.2 Data Collection and Analysis

Data collection methods classified as first- and third-degree are considered. A third-degree method is the analysis of existing artifacts available in repositories that have been created in advance of the case study. Data sources may be public documentation and literature, meta-information in the version control system, or metrics from the source code via static code analysis. Interviews are a first-degree method of data collection and provide more control over the type and quality of the collected data as compared to the previous method (Runeson et al., 2012c).

The methods are employed in the following order to approach the investigation from contextual data, over source code metrics to the developers' perspectives:

1. Analyze the available literature and documentation to obtain qualitative data on the project's origin, environment, and problem domain.
2. Recover the as-is SA from the source code via SAR to apply metrics and extract other quantitative data to find possible indicators for AE via SAA.
3. Conduct qualitative interviews with a selection of contributors about their perception of AE and its symptoms to learn about its awareness in the project.

One objective of using multiple types of sources of information is to gain a holistic overview of the case from different perspectives. For example, the source code itself may reveal indicators for AE that are linked to organizational issues and have persisted since the beginning of the project or have occurred due to social factors, such as onboarding and offboarding of contributors, which was less of a concern in earlier periods. Therefore, a portrait of the original environment, research background, and software design decisions is necessary to understand aspects of the evolution of the project.

3.2.1 Literature and Documentation

JackTrip, as an OSS originating from academia, has been described in several scientific publications for its software design and applications. Therefore, information regarding its original approach in the problem domain, development, and deployment is most probably contained in such sources. This may serve as a foundation for understanding its early evolution to some extent. Additionally, in 2015, Nelson wrote the book *The*

Sound of Innovation: Stanford and the Computer Music Revolution, which describes how the academic institute, in which JackTrip has been originally developed, was founded. It provides background on the academic environment for further investigation.

To find papers addressing software design and implementation, the Publications sub-page of the *SoundWIRE Research Group* website lists many publications up to 2010 with different foci. The documents were skimmed and evaluated for their relation to the aspects of SA resulting in the following choice of titles (CCRMA - Stanford University, 2010):

- "JackTrip/SoundWIRE Meets Server Farm" (2010),
- "JackTrip: Under the Hood of an Engine for Network Audio" (2010),
- "Levels of Temporal Resolution in Sonification of Network Performance" (2001),
- "A Simplified Approach to High Quality Music and Sound Over IP" (2000).

These publications constitute a basis for understanding the problem domain and how it is approached. Additionally, a more recent paper with the title "I am Streaming in a Room" (2018) is analyzed as it is included in the `papers` folder of the git repository and can be considered a point of reference in the project. Newer papers were found with the Bielefeld Academic Search Engine (https://www.base-search.net) using the search string `jacktrip` in combination with a date filter keyword `year:[2011 TO 2022]`[15] and applying the same reading technique previously mentioned to identify relevant sources. As most of the documents describe application scenarios without technical focus, the article "Experiencing Remote Classical Music Performance Over Long Distance: A JackTrip Concert Between Two Continents During the Pandemic" (2021) seems to be the only newer source of relevant contextual information on the JackTrip project.

Ultimately, the official project documentation referred to in the git repository is used for gathering current details and organizational aspects of the project. That information may be necessary to interpret and link other data collected from source code, for example.

15 https://www.base-search.net/Search/Results?type=all&lookfor=jacktrip+year%3A%5B2011+TO+2022%5D

3.2.2 Source Code

For recovering the actual SA from implementation files the Java software an ***A**rchitecture **R**ecovery, **C**hange, **A**nd **D**ecay **E**valuator (*ARCADE*)* is used. It provides recovery methods, as described in 2.3.5 Architecture Recovery Techniques, for creating different architectural views on a system and offers functions for generating visualizations from the recovered architectures and applying metrics to them. Additionally, architectural views can be analyzed for smells (Schmitt Laser et al., 2020). The original project available on BitBucket is unmaintained, as the primary author Marcelo Schmitt Laser started a reimplementation called ARCADE Core, which is located at GitHub[16]. Thus, ARCADE Core is employed to recover the SA of JackTrip using ACDC and PKG techniques. Its requirements are a running Perl installation and a Java runtime environment version 11 or above. The codeblocks for documenting the command-line interface (CLI) instructions in the following sections are described for Unix-like operating systems.

```
arcade_core
└── subject_system
    ├── clusters
    │   ├── acdc
    │   └── pkg
    ├── deps
    ├── graphs
    │   ├── arch
    │   │   ├── acdc
    │   │   └── pkg
    │   └── clusters
    │       ├── acdc
    │       └── pkg
    ├── metrics
    │   ├── acdc
    │   └── pkg
    ├── smells
    │   ├── acdc
    │   └── pkg
    └── versions
```

Codeblock 1: Directory structure for the used ARCADE Core workspace

The following steps assume a directory structure as illustrated in Codeblock 1 to store the executable files from the ARCADE Core release in its root and JackTrip's source code and analysis data inside a `subject_system` subdirectory.

16 https://github.com/usc-softarch/arcade_core

Preparations

First, the version control system is used to clone the source code branch for each release[17] of JackTrip in the `subject_system/versions` directory and rename its folder to match the convention `projectname-x.y.z`, where `x`, `y`, and `z` reflect major, minor, and patch version numbers. Codeblock 2 shows example commands for version 1.6.8. To reduce the number of systems, release candidates (`-rc`) are not included.

```
git clone -b v1.6.8 https://github.com/jacktrip/jacktrip

mv jacktrip jacktrip-1.6.8
```

Codeblock 2: Example of cloning and renaming a JackTrip release branch

As the folder hierarchy is regarded as structural information in some recovery methods and is currently not checked by the ARCADE Core tool, the directory structure in release v1.1 of JackTrip needs to be adapted manually. The files and directories inside the additional `jacktrip` subfolder need to be moved to the upper directory root of the project. Otherwise, any content in the additional subfolder is treated as a different entity in the clustering process of ARCADE Core, and it causes distortion in the results.

For analyzing C-based systems with ACDC and PKG recovery only the following assets need to be downloaded from the release page into the root of the `arcade_core` directory (Schmitt Laser, 2022):

- `ARCADE_Core.jar`
- `mkdep.pl`
- `mkfiles.pl`

The `ARCADE_Core.jar` binary in the latest release v1.2.0 from November 12th, 2022, provides the most basic functionality and was used at first. However, some necessary features regarding recovery and metrics haven not yet been implemented. For that reason, the author uploaded a prerelease[18] saved as `ARCADE_Core_prerelease.jar` and referred to in the `-cp` parameter of several commands. Apart from that particularity, the next steps basically follow the instructions on the release page.

17 https://github.com/jacktrip/jacktrip/releases
18 https://drive.google.com/file/d/17itiZyv41mCmA5Ji2dkts1zkN7O9Wvb1/view

Fact Extraction

Before architectural views can be generated, the actual dependencies in the source code need to be extracted. The responsible ARCADE Core function is known as `CSource-ToDepsBuilder` and uses the Perl scripts `mkdeps.pl` and `mkfiles.pl`.

```
for version in subject_system/versions/*
do
  java -cp ARCADE_Core.jar \
  edu.usc.softarch.arcade.facts.dependencies.CSourceToDepsBuilder \
  $version \
  subject_system/deps/$(basename $version)_deps.rsf \
  subject_system/deps/$(basename $version)_deps.json
done
```

Codeblock 3: Bash for-loop to run the fact extractor on each version

It processes the system source as input (first parameter) and must be invoked on each version individually, making it feasable for a bash `for`-loop to iterate through the `ver-sions` directory (Codeblock 3). The second and third parameter define the output as a human-readable list of the system dependencies in Rigi standard format[19] (RSF) and a JSON file containing serialized feature vectors to be placed in `subject_system/deps`.

Clustering

Using the dependencies previously extracted from implementation files, the ACDC clusterer in ARCADE is invoked by providing the system dependencies RSF file as input and another RSF file for storing the clustering information output. As shown in Codeblock 4, each system version must be processed individually using a for-loop.

```
for version in subject_system/deps/*.rsf
do
  java -cp ARCADE_Core.jar \
  edu.usc.softarch.arcade.clustering.acdc.ACDC \
  $version \
  subject_system/clusters/acdc/$(basename $version)_clusters.rsf
done
```

Codeblock 4: ACDC clustering with RSF dependencies files input

The output is written to the path `subject_system/clusters/acdc`, which contains a single cluster RSF file per version. It is used as an architectural view in later steps.

19 https://rigi.cs.uvic.ca/downloads/rigi/doc/node52.html

For PKG recovery, the dependencies file for each system version is also processed. In ARCADE Core the `edu.usc.softarch.arcade.clustering.Pkg` function expects five parameters and outputs RSF files into the directory `subject_system/clusters/pkg`, when using the for-loop shown in Codeblock 5.

```
for version in subject_system/deps/*.rsf
do
  java -cp ARCADE_Core_prerelease.jar \
  edu.usc.softarch.arcade.clustering.Pkg \
  projectname=jacktrip \
  projectversion=$(basename $version | cut -d '-' -f 2) \
  projectpath=subject_system/clusters/pkg \
  depspath=$version \
  language=c
done
```

Codeblock 5: Clustering using the PKG recovery method

The parameters are described as follows:

- `projectname`: The name of the project is used as a fragment in the output file.
- `projectversion`: Additionally, the version is used as a fragment for the output filename. In the codeblock above, it is provided by a pipe of the two commands, `basename` and `cut`, to strip the version from the RSF file contained in `$version`.
- `projectpath`: The output files are written to this relative or absolute path.
- `depspath`: The dependencies RSF file for the system version is provided as input for the PKG clustering function.
- `language=c`: The language parameter indicates a C-based system as input.

As the PKG recovery function applies a different naming convention that is not explicitly considered in the following codeblocks, the RSF files are manually renamed to match the pattern `jacktrip-<version>_clusters.rsf`.

Smell Detection, Metrics and Visualizations

By building the architectural views with different recovery techniques in the previous section, subsequent smell detection, measurements, and visualization can be performed. The following commands are executed for both recovery techniques: Codeblocks serve as examples for the appropriate order and specification of the parameters for either ACDC or PKG dependencies as input. For replication, they must be adapted to generate the data for ACDC or PKG, respectively.

Codeblock 6 shows the invocation of the `ArchSmellDetector` function that creates a JSON file for individual versions recovered by ACDC. These contain an allocation of clusters for a specific type of smell, which is saved in `subject_system/smells/acdc`.

```
for version in subject_system/deps/*.rsf
do
  java -cp ARCADE_Core.jar \
  edu.usc.softarch.arcade.antipattern.detection.ArchSmellDetector \
  $version \
  subject_system/clusters/acdc/$(basename $version)_clusters.rsf \
  subject_system/smells/acdc/$(basename $version)_smells.json
done
```

Codeblock 6: Smell detection for individual versions recovered by ACDC

As the data inside the JSON file is stored without line breaks and indentation, it needs to be converted with, for example jq[20], to be human-readable, as shown in Codeblock 7.

```
for version in subject_system/smells/acdc/*.json
do
  jq -S . $version > subject_system/smells/acdc/$(basename -s .json \
  $version)_human-readable.json
done
```

Codeblock 7: Formatting conversion of smells JSON files of ACDC recovery

Codeblock 8 is an example of the content. A `smellcollection` consists of a `clusters` array carrying involved clusters, a `topicNum`, and a `type` that defines the type of smell. According to the original ARCADE manual, `bdc` stands for dependency cycle and `buo` stands for link overload. Due to the chosen recovery methods, only those smell types are detected (Computer Science Department University of Southern California, 2017).

```
{
  "smellcollection": [
    {
      "clusters": [
        "src/jacktrip_globals.ch",
        "src/main.cpp"
      ],
      "topicNum": -1,
      "type": "buo"
    }
  ]
}
```

Codeblock 8: Example content of a human-readable smells JSON file

20 https://stedolan.github.io/jq

Using `grep` with the `-c` parameter in another bash for-loop, the number of each smell type can be counted for every system version and written to a CSV file to create a graph with spreadsheet software (Codeblock 9).

```
echo "Version, Dependency Cycle, Link Overload" > \
subject_system/smells/acdc/smellstrend.csv

for version in subject_system/smells/acdc/*human-readable.json
do
  echo "$(basename $version | cut -d '-' -f 2 | cut -d '_' -f 1), \
  $(grep -c 'bdc' $version), $(grep -c 'buo' $version)" \
  >> subject_system/smells/acdc/smellstrend.csv
done
```

Codeblock 9: Counting smells per system version saved into a CSV file

The generation of architectural metrics is carried out by specifying the super directory of the clusters as `systemdirpath` and the super directory of the dependencies list as `depsdirpath` in the `SystemMetrics` function, as shown for PKG in Codeblock 10.

```
java -cp ARCADE_Core_prerelease.jar \
edu.usc.softarch.arcade.metrics.SystemMetrics \
systemdirpath=subject_system/clusters/pkg \
depsdirpath=subject_system/deps \
outputpath=subject_system/metrics/pkg
```

Codeblock 10: Architectural metrics generation for an existing PKG view

The output is written to several CSV files to create graphs. Other metrics of high interest are the *architectural instability*, Basic- and TurboMQ, which are aggregated in `versionMetrics.csv`. According to the author of ARCADE Core the architecture instability is the average of the clusters', which is written to the file `instability.csv`. However, cluster instability only considers edges that are external to the cluster, meaning edges between an entity inside the cluster and an entity in another cluster. Fan-in (afferent coupling) is calculated as dependencies from other clusters to the one being measured, and the reverse applies for fan-out (efferent coupling). Both are written to CSV files named `fanIn.csv` and `fanOut.csv`, respectively. Therefore, it does not directly relate to the instability of the entities inside the cluster. It is meant to be a separate measure of architectural instability. BasicMQ is calculated and aggregated in `basicmq.csv` beforehand, and the files `a2a.csv` and `cvg.csv` provide a2a and cvg metrics. They are described in the System-level Metrics section of subchapter 2.3.6.

ARCADE Core can convert RSF files to DOT format to create visualizations of the system version dependencies by using the function `RsfToDot` on the recovered clusters. It expects the mode (`arch` or `cluster`), the dependencies RSF file, the cluster RSF file, and an output file (Codeblock 11).

```
for version in subject_system/deps/*.rsf
do
  java -cp ARCADE_Core_prerelease.jar \
  edu.usc.softarch.arcade.clustering.RsfToDot \
  arch \
  $version \
  subject_system/clusters/pkg/$(basename $version)_clusters.rsf \
  subject_system/graphs/arch/pkg/$(basename $version)_graph.dot
done
```

Codeblock 11: Visualization of the architecture recovered by PKG

The command in Codeblock 11 is adapted for a second invocation, in which the first parameter `arch` is changed to `cluster` in order to create a RSF file for every single cluster to be placed inside a directory instead of a single file. Therefore, the last parameter is altered to: `subject_system/graphs/arch/pkg/$(basename $version)_graph.dot`.

Finally, the conversion to SVG format is performed by GraphViz[21] and automated by iterating through the generated DOT files for each version. For the clusters, a nested for-loop is necessary, as shown in Codeblock 12 as an example for ACDC clusters.

```
for version in subject_system/graphs/clusters/acdc/*
do
  for cluster in $(ls $version)
  do
    dot -Ksfdp -Tsvg $version/$cluster > $version/$cluster.svg
  done
done
```

Codeblock 12: Conversion of DOT files to SVG via GraphViz dot utility

At this point, the workspace in `subject_system` is populated by many files that can be used for visualization and reporting of metrics for the recovered architectural views generated by ACDC and PKG. However, metrics and smell detection, such as those of ARCADE Core, can only indicate a probability of eventually having issues and must be supported or contradicted by other sources of evidence such as developer interviews.

21 https://graphviz.org

3.2.3 Interviews

The Interview sessions were conducted via videocall from January 21st and February 2nd, 2023. The participants were primarily chosen depending on their different involvements in the project (academic research, profession, hobby). Four contributors were invited and are characterized as follows:

Interviewee #1: Academic contributor since 2000

Interviewee #2: Hobbyist contributor since 2019

Interviewee #3: Professional contributor since 2020

Interviewee #4: Academic and hobbyist contributor since 2020.

By using open questions and a semi structure, participants in the interviews were allowed to elaborate on the topics. As the order of questions was not planned and depended on the way the interviews evolved, a guideline was necessary to remain focused and replicate the procedure (Runeson et al., 2012c). The structure consisted of an introduction, entry questions, main questions, and a retrospective. In the introduction, the greeting and acknowledgement of taking the time began the session. A brief description of the interview structure and its duration was given, followed by asking for agreement that the sessions be recorded and used for research as explained in a privacy agreement document provided in advance. If the interviewee had no questions in this regard, a short overview on the topic of SA and AE prepared for two entry questions, one with follow-up, or spontaneous questions depending on the interaction with the interviewee:

- How long and for what motivation are you contributing to JackTrip?
- Please tell me about challenges in project organization, processes, and tooling but also with technology adoption and infrastructure, for example.
    - How does the on/offboarding of developers happen?
    - What could be the barriers to engagement?

The main questions included spontaneous questions that focued on topics regarding the perception of AE and possible reasons for it according to literature on the theoretical background introduced subchapter 2.3.4, as follows:

- Consideration of artistic and research functionalities during development
- Special approaches due to the problem domain
- Change in the speed of evolution compared to previous periods and its effects
- Mistakes due to frequent changes and strategy for the overall picture
- Time constraints influencing development work.
    - Phase and reason for time pressure during development
    - Workload change and coping with it
- Design decisions that have rendered the system difficult to maintain or evolve
- Consideration of cohesion and coupling in discussions.
- Imagining the differences if the project today began today.

Finally, another acknowledgement for taking the time finished the interview. Subsequent steps, such as transcription and revision by the interviewee, are explained before a farewell ended the interview session.

The interview recordings were automatically transcribed using OpenAI Whisper[22] as described in the setup guide. The `medium.en` settings were specified as language model. Manual correction by the interviewer and revision by the interviewees allowed quality control. The final Interview Transcriptions can be found in Chapter 12.1.

After transcription the interviews were examined to identify topics, ideas, and patterns for summarization into broader themes using a *thematic analysis*. This is a flexible approach to analyzing views, opinions, knowledge, experiences, and values from a set of qualitative data such as interview transcripts. To use the outcome in combination with other data collected before and relate it to the theoretical background, a deductive and semantical approach was chosen, which involved the analysis of the explicit content of the data. Predefined themes were derived from the research questions and included in the topics of the interview guideline. They are specified as follows: rushed evolution, recurring changes, time pressure, accumulation of design decisions, lack of developers' experience, academic origin, and problem domain specifics (Caulfield, 2022).

22 https://github.com/openai/whisper

Thematic analysis is a six-step process comprising familiarization, coding, generating themes, reviewing themes, defining and naming themes, and writing up. An overview of the collected data is an important starting point to become familiar with it. This is accomplished by carefully studying the texts and taking initial notes. Then, parts of text, — sentences and phrases — are highlighted and labeled with a code describing their content. Everything that might be interesting is marked while going through each transcription. At the end the data from the texts are collated into groups to outline recurring main aspects and meanings. In the next step, the codes are examined to identify patterns for summarization into broader themes. Vague or irrelevant codes may be discarded, and codes can become themes. As the presentation of developers' perceptions of AE and its symptoms is the purpose of the analysis, it is necessary to build themes of particular interest regarding this. Quality control is carried out by reviewing the themes and comparing them to the data. If necessary, combination, splitting, dismissal, and creation of new themes can be performed before defining and naming them. This step supports an easier understanding of the data and aims to find a succinct name for each theme. Finally, this work represents the writing-up step, including the description of each theme with examples from the data as evidence, its frequency of appearance among the interviews, and its relation to other data collected by other means (Caulfield, 2022).

3.3 Threats to Validity

Threats to the validity of this work are discussed by following the guidelines proposed by Runeson et al. (2012b). This classification scheme is also used by Yin (2018). Instead of changing terms, the four aspects of validity—construct validity, internal validity, external validity, and reliability—known from other research methodologies are operationalized for a flexible design study.

3.3.1 Construct Validity

Threats to construct validity describe whether the measurements and findings answer the research questions and whether appropriate methods have been chosen. One concern is that the literature and documentation about the JackTrip project was created for different purposes and might not include all the necessary information. Therefore, data validity and completeness cannot be guaranteed as there is no control over the quality of

the data. This threat is mitigated by using two other data sources (source code and interviews) to gain another perspective and identify contradictions between them, which is also known as data source triangulation. Another threat is related to the individual limitations of single recovery methods. As each method provides a distinct architectural view, the recovered architecture is biased. Consequently, any measurement or smell detection applied to that view is also dependent on it. To be able to compare the results, multiple recovery methods are employed.

3.3.2 Internal Validity

Internal validity is threatened by environmental influences on the researched phenomenon during the conduct of the study at hand. A strategy for elimination is the limited selection of JackTrip versions used in SAR and SAA. These system versions have been released prior to investigation and checked for representativeness in advance. Concerning the internal validity of the data sourced in the semistructured interviews, a guideline with predefined questions is used to allow focus and replication of the procedure.

3.3.3 External Validity

External validity concerns the extent to which the results can be generalized. Due to the nature of a case study there is a legitimate concern regarding nongeneralization. However, by providing contextual information on the case, it is possible for the reader to understand its relevance for similar situations, as described by Runeson et al. (2012b).

3.3.4 Reliability

In order to improve the reliability of the study, which is the extent of the dependency of the data and analysis on the specific researcher, each step in this research is documented and explained in detail. For example, the usage of ARCADE Core is supported by providing the used commands in codeblocks, and full interview transcriptions are found in the appendix of this work. Regarding ARCADE Core, there is a certain threat of nonreplicability as the tool is still in active development and has not had another official release; this includes the features used for metrics reporting and visualizations. Fortunately, its current developer is reachable and responds quickly to any support inquiry.

4 Results

The results comprise a number of different pieces of data that are presented individually. For orientation, this chapter is structured in the same order as the single sources of evidence, and their analysis methodologies as introduced in subchapter 3.2 Data Collection and Analysis:

1. Literature and documentation analysis,
2. Software architecture recovery and analysis, and
3. Thematic analysis of interviews

As a result, a traceable chain of evidence can be established for discussion in Chapter 5.

4.1 Literature and Documentation Analysis

The chosen literature and documentation explain JackTrip's academic origin, its project history, the evolution of operation modes, and the state of current documentation. By reporting this contextual information, the case for this study is introduced.

4.1.1 Academic Background and Environment

As an institute at the intersection of music and computer science, the Center for Computer Research in Music and Acoustics (CCRMA, pronounced "karma") at Stanford University has a long tradition of multidisciplinary projects unveiling new scientific and artistic fields of endeavor while developing key technologies for commercial applications. This tradition harks back to its roots in the 1960s, when John Chowning and other pioneers invented frequency modulation (FM) synthesis. As a composer, he collaborated with engineers, psychologists, computer scientists, and other musicians seeking novel ways to produce and manipulate sound using a computer. In order to raise funding to establish an institute for further research, the FM synthesis was later licensed to the Yamaha Corporation of Japan to drive the DX7 synthesizer, which became the most successful digital synthesizer instrument. That cooperation was part of CCRMA's struggle to constantly fight for legitimacy and tenured faculty while developing groundbreaking technology (Nelson, 2015).

Though controversies about sharing knowledge exclusively with a foreign company and frequent license negotiations with the Yamaha corporation trying to protect intellectual property for commercial reasons threatened to trap CCRMA in license politics, it managed to reflect back on the code-sharing culture it once coined and join the open source movement before the millennium. In 1996, Fernando López-Lezcano, a lecturer and system administrator, experimented with using GNU/Linux as an operating system platform for machines running CCRMA software with regard to whether audio interfaces would be supported and how latency could be effectively reduced (Nelson, 2015):

> Patches became available for the Linux kernel that enabled it to start working at the low latencies suitable for realtime reliable audio work, so I started building custom monolithic kernels that incorporated those patches and all the drivers I needed for the hardware included in our machines. (Nelson, 2015)

With a growing number of machines, standardization became necessary. As CCRMA users wanted to run the same software on their computers at home, López-Lezcano began compiling packages for distribution and writing instructions on how to install and configure a GNU/Linux audio system. This soon evolved into a repository of free audio software packages hosted on a website called Planet CCRMA, which enables access to worldwide users outside of the university network. Outside users benefited from receiving cutting-edge software, and the authors of audio software at CCRMA gained feedback on applications outside academia. Thus, a breadth of different backgrounds and environments contributed to a shared resource, forming a community of users (downstream) and research (upstream), according to Nelson (2015).

The phenomenon that people composing music also happen to develop software is not a coincidence as “both activities rely on symbolic representations that are removed from the ultimate execution or performance” (Nelson, 2015). Referring to an interview with Chowning, musicians are used to score-writing on paper and cannot listen to resulting sounds while approaching. Additionally, the process of coding and writing music can be similar in terms of aesthetic aspects. Ge Wang, an assistant professor at CCRMA and cofounder of Smule who has also been interviewed by Nelson, stated:

> [W]riting music and writing software are very much alike. They both, for me at least, involve a lot of tinkering. […] Then, there are all these little

> rewards along the way when you get a passage working or you get a little feature or something working in the software and then that allows you to see more of what the final product is going to be [...] (Nelson, 2015).

Regarding such connections, Nelson stresses the term “multivocality” in describing how different backgrounds allow different interpretations of a common technology, forming new disciplines at their intersection. Chris Chafe, the current director of CCRMA, combines his listening experience as a musician with his technical knowledge of sound synthesis for sonifying delays in networks and realizing distributed music performances over the Internet implemented as an OSS as described in the following (Nelson, 2015).

4.1.2 Project History

Early studies on Internet acoustics at CCRMA revealed the need for a system capable of streaming uncompressed audio over the internet protocol (IP) suite with low latency. Around the year 2000 Chris Chafe founded the SoundWIRE (Sound Waves from the Internet from Real-Time Echoes*)* group[23] to research network delays and their effect on interactive audio transmission to evaluate networked music performance (NMP).

SoundPing and SoundWIRE

The SoundPing tool was created to sensorially assess the quality of service (QoS) of network connections with regard to jitter and packet loss. Jitter describes varying transmission times of packets due to congestion or different paths irritating time-critical periodic signals. Packet loss is another downside of packet-switched networks also leading to audible artifacts. As the ear is highly sensible to temporal changes such as pitch, mapping round-trip time (RTT) data to tones for listening to the QoS of a connection appealed intuitively to Chafe and Leistikow (2001). Round-trip time, as reported by the Ping utility, reflects delay times of networks in milliseconds. SoundPing serves as parameterized display extending SoundWIRE and was developed in advance for sound generation via physical modeling synthesis[24] and streaming using the unified datagram protocol (UDP). SoundWIRE's basic idea is to use a network connection as an acoustical medium that resonates with a trigger signal of noise bursts to characterize its QoS. This is comparable to playing a stretched guitar string that is detuning or interrupting in

23 https://ccrma.stanford.edu/groups/soundwire/
24 https://ccrma.stanford.edu/software/clm/compmus/clm-tutorials/pm.html

relation to jitter or packet loss, respectively. Literally, a signal is plucking the network using the Karplus-Strong string synthesis (physical modeling) algorithm and the network latency as a delay to produce sounds. This technology is not only intended for technical, but also for artistic or musical application. A more recent paper explains that the aforementioned algorithm was later replaced by *Freeverb*[25] and implemented with a new digital signal processing programming language called *Faust*[26] that facilitates "generating complex multichannel circuits which can be emitted as C++ and then compiled into the ProcessPlugin format" (Chafe, 2018).

However, around 2000, SoundWIRE leveraged a client/server model allowing three operational modes—loopback, unidirectional, and bidirectional— for *Peer-to-Peer* (P2P) setups. It comprises a core stream class with plugins for input, output, processing, and input mixing. Input and output plugins use their own threads, calling for data from or to the core class. The streaming engine was built against the *CommonC++*[27] library for thread and socket support and *Qt*[28]. Additionally, the Synthesis Tool-Kit[29] (STK) served for processing tasks during prototyping. For example, a reverb could be applied to alter the signal as it passed through the stream. The mixing of signals was another crucial feature for musical application as there was no additional abstraction layer, such as a sound server, available at that time. From the application side of things, first remote recording sessions and teleconcerts in cooperation with McGill University served as proof of concept and showed potential in addition to compositions leveraging SoundPing's mapping features for the sonification of historical RTT data (Chafe et al., 2000).

JACKyfication

In the following years, more experiments and performances drove further development, and the project was eventually renamed to *JackTrip* due to the integration of the Jack Audio Connection Kit (JACK)[30] sound server allowing Mac OS X portability and multicore utilization through the work of Juan-Pablo Cáceres and Chris Chafe, in 2009. One thread processes audio in- and output through JACK, and a separate sender or receiver

25 https://ccrma.stanford.edu/~jos/smith-nam/Freeverb.html
26 https://faust.grame.fr
27 https://www.gnu.org/software/commoncpp
28 https://www.qt.io
29 https://ccrma.stanford.edu/software/stk
30 https://jackaudio.org

thread packs/unpacks that audio data into datagrams for UDP transmission or reception, respectively. Arbitrating between those threads is a ring buffer that reduces processing latency by sending only actual audio (not silence) and avoiding glitches at the receiver side. Figure 10 shows the interaction of the ring buffer and threads.

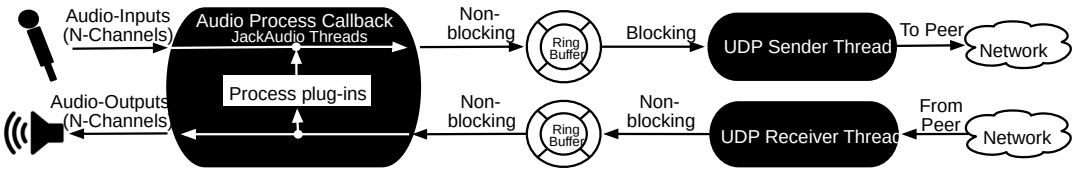

Figure 10: JackTrip threading architecture by Cáceres and Chafe (2010b)

Though realtime scheduling[31] of JackAudio and socket threads facilitates low latency locally, maintaining a stable connection over UDP would require a great deal of buffering, thus leading to high latencies. By the nature of UDP (connectionless communication without retransmission and error-correction) and by using a constant low buffer size, buffer glitches are possible due to over- and under-run conditions at the receiver. To account for this, the strategy is to either output zeros (silence) or loop the most recent audio data (wavetable synthesizer[32]-alike) without resetting the ring buffer read pointer. In case of a buffer under-run, the write pointer is reset to compensate for clock drifts that originally caused both suboptimal buffer conditions (Cáceres & Chafe, 2010a).

Entering the Cloud

Since the summer of 2009, multilocation concerts have evinced the need for a central audio mixing and relaying feature that requires to implementation of a multi-client concurrent server design called *hub mode*, as described by Cáceres and Chafe (2010a). In this mode, shown in Figure 11, the server listens on port 4464 for a client connection via TCP. It reads the IP address and port in packets to check for known connections. If the connection is unknown, a new port is allocated for running the actual UDP communication with the connecting client as a clone of the JackTrip worker thread, which is added to a thread pool. Additionally, the new connection characteristics (IP address and port) are saved to an array to allow the previously mentioned probing.

31 https://wiki.linuxfoundation.org/realtime/start
32 https://ccrma.stanford.edu/~jos/kna/Taxonomy_Digital_Synthesis_Techniques.html

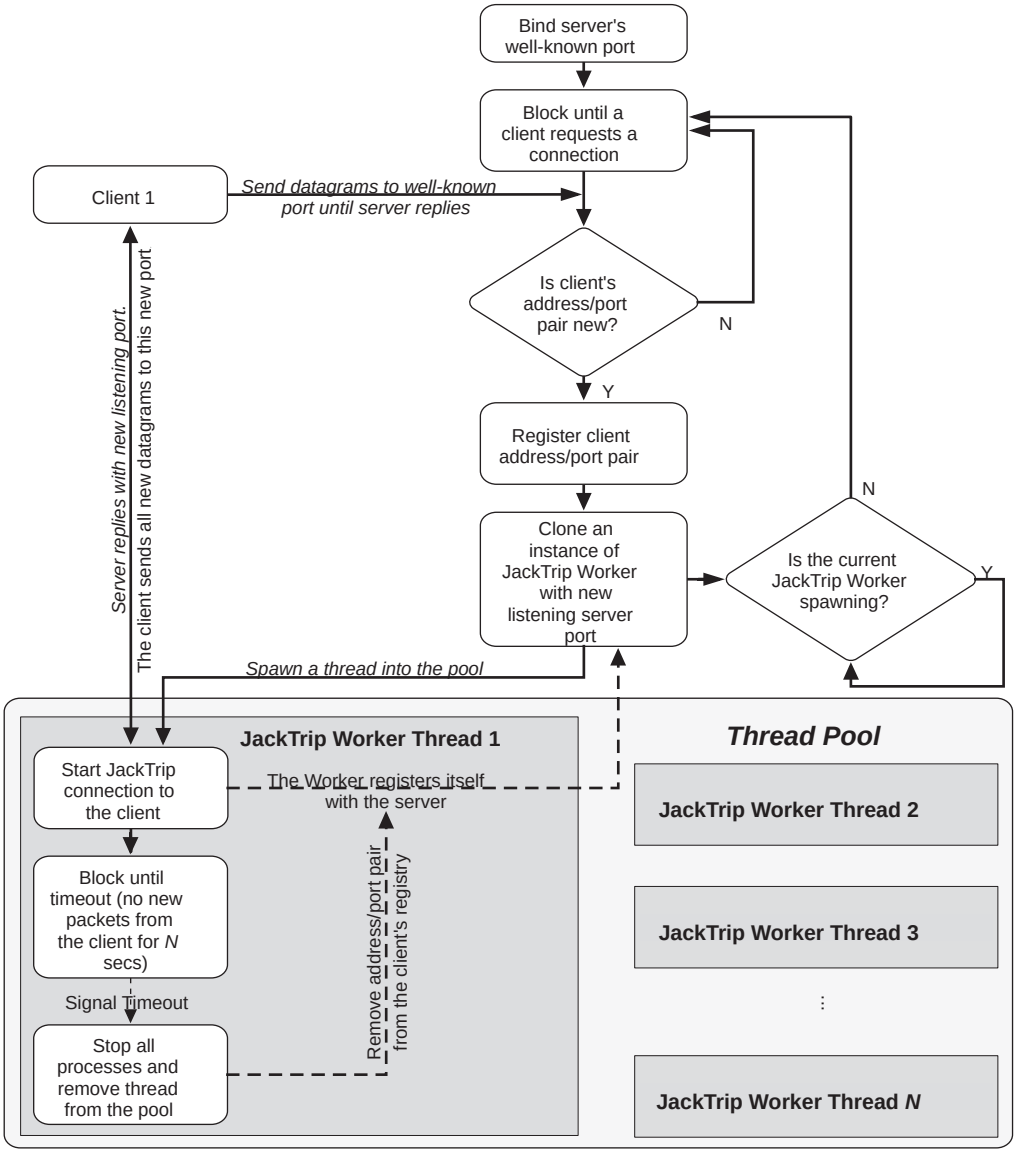

Figure 11: JackTrip's multi-client concurrent server design (Cáceres & Chafe 2010a)

Afterwards, the server can manage new clients in the same way. Client connections are removed after a client has stopped sending packets or the server has not received packets for a certain time. JackTrip server audio mixing and rerouting capabilities allow dynamically connecting client audio with an arbitrary number of client channels at the cost of a fixed sample rate and buffer size due to JACK limitations but with benefits regarding central gain control and interconnection of streams (Cáceres & Chafe, 2010b).

Upon streaming, QoS is achieved using the recirculation method developed during the SoundWIRE period. This enables the user to acoustically fine-tune the JackTrip instrument to a high pitch while trading off latency and audio quality.

> As the network delay increases, the pitch of the st[r]ing will be lower. Variances in the latency will be perceived as vibrato of the string model. Packet losses are translated into impulsive types of sounds (for the case when the receiver plays zeros when it doesn't receive a packet) or into wavetable types of sounds (for the mode when the system keeps looping through the last packet received) (Cáceres & Chafe, 2010b).

Cáceres and Chafe (2010a) kept in mind that JackTrip is supposed to be used by musicians rather than engineers.

Recent Application

Since JACKyfication in 2009 and scaling via roll-out on a server farm (cloud) in 2010, many concerts and NMPs have been realized, while similar systems have evolved and other projects have integrated JackTrip or extended its features. For example, a new graphical user interface (GUI) called *QJackTrip*[33], dedicated solutions such as Raspberry Pi-based systems[34] as well as Pure Data and WebRTC implementations[35] emerged in short time. Lockdowns at the beginning of the COVID-19 pandemic favored NMPs over the Internet for music classes and international orchestras. This has driven the development of JackTrip as videoconferencing software neglects low-latency audio. In 2020 the JackTrip Foundation[36] and JackTrip Labs, Incorporated[37] were founded for software development, training, and support (Bosi et al., 2021). The JackTrip OSS is sustained by the nonprofit JackTrip foundation and utilized by JackTrip Labs in a business model offering services and infrastructure. Virtual Studio sessions leverage the cloud infrastructure of JackTrip Labs as rentable studio minutes to be used either by running the cross-platform OSS on an existing computer with professional audio gear or by purchasing a standalone JackTrip bridge device based on Raspberry Pi. The JackTrip OSS is a centerpiece for these applications (JackTrip Labs, Incorporated, 2022).

33 https://www.psi-borg.org/other-dev.html
34 https://hvc.berlin/projects/sprawl_system
35 https://github.com/jacktrip-webrtc/jacktrip-webrtc
36 https://www.jacktrip.org
37 https://www.jacktrip.com

4.1.3 Developer Documentation

One source of information about the evolution is the Releases page on GitHub, which provides dates of released JackTrip versions with information about included changes, for exmaple the first experimental introduction of the hub mode server in version 1.1, IPv6 compatibility in version 1.2, and Virtual Studio integration in version 1.6.0. For a visual overview, Figure 12 shows a timeline that illustrates the distance between minor releases and the accumulation of released patch versions as of November 2021.

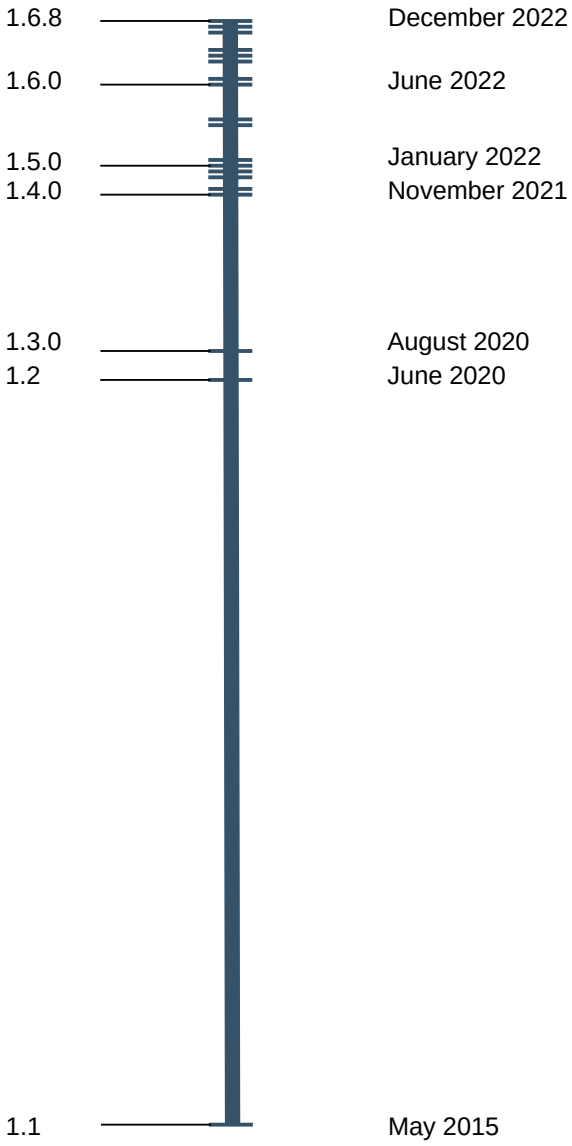

Figure 12: JackTrip’s releases timeline from versions 1.1 to 1.6.8

The current documentation of the project is preserved as a static website via GitHub Pages (at https://jacktrip.github.io/jacktrip) and built from markdown files in the `docs` subdirectory of the git repository using the MkDocs tool. It includes a user guide and compilation instructions for widespread GNU/Linux distributions (Ubuntu and Fedora), mac OS and Windows using QMake[38], or alternatively, the Meson build system[39]. Dependencies are a C++ compiler and Qt5, a recent version of the library previously mentioned in the SoundPing and SoundWIRE section. Optionally RtAudio[40] can be used as audio backend (Tonnätt, 2021a).

Table 3: Changelog items for the releases 1.5.3 and 1.6.0

Release	Changelog items
1.6.0	(added) Virtual Studio integration; previous GUI is now called "Classic Mode" (added) dblsqd for auto-updates (updated) buffer strategy 3 - multiple updates and fixes, still experimental (added) JackTrip Labs signing scripts (fixed) OpenSSL in the build script (updated) code cleanup and maintenance
1.5.3	(added) linux instructions for parallel versions (added) docs on running JackTrip with a named JACK server (added) nogui linux release build (update) Auto mode for buffer strategy 3 (update) remove extra macOS binary release artifact (fixed) Don't link nogui qmake build with gui libraries

In Table 3 the changelog items for versions 1.5.3 and 1.6.0 are listed as examples of information that is communicated in the Changelog page. It contains some details on the Virtual Studio introduction. A Resources page explains that the source code has been migrated several times. Version 1.0 started at Google Code and migrated to GitHub for the first time with versions 1.05 to 1.1. It has been there again since the spring of 2020 after being moved temporarily to CCRMA's cm-gitlab. In the Links section, an outdated API description that has been automatically generated from source code with Doxygen, is referred to and other locations of information about the project created throughout its evolution are listed. These historical guides and discussions are primarily written for users and are thus not considered in the following (Tonnätt, 2021b). The next subchapter deals with source code metrics identified by SAR and SAA in the evolution of the OSS.

38 https://wiki.qt.io/Qmake
39 https://mesonbuild.com
40 https://www.music.mcgill.ca/~gary/rtaudio

4.2 Software Architecture Recovery and Analysis

In order to gain an overview of the architectural views recovered by ARCADE Core, as explained in subchapter 3.2.2, the report of findings is structured as follows:

1. Trends of Architectural Metrics
2. Trends of Architectural Smell Types
3. Visualization of Selected Architectures and Clusters

The objective is to identify noticeable distortions in the graphs to narrow the scope and gain an overview of issues in the project to support the selection of representative visualizations. Using this approach, certain occurrences are locatable inside individual components of the system's versions based on the views of the recovery methods.

4.2.1 Trends of Architectural Metrics

Metrics, as described in subchapter 2.3.6 Architectural Metrics, reveal the trends of architectural changes through successive versions. The graphs in the following sections illustrate their evolution for the ACDC recovery method and for the PKG recovery method, respectively. In Figures 13 and 14, each architecture of a system version is compared with 1.6.8 using the a2a metric. The figures show the trend for both recovery methods, which are nearly identical. For that reason, the course of the values is only described once in the following:

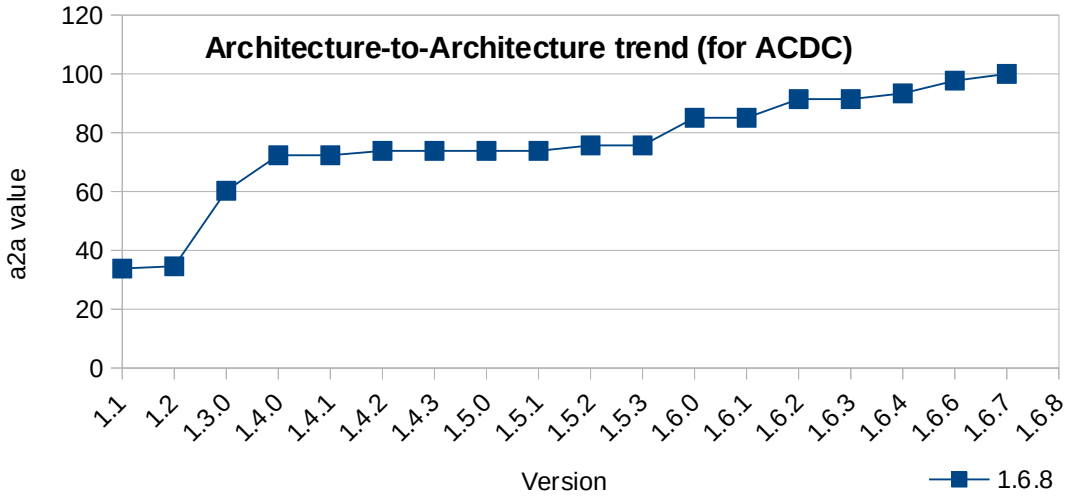

Figure 13: a2a trend for ACDC recovery related to JackTrip version 1.6.8

In Figure 13, the similarity of versions 1.1 and 1.2 is below 35% and significantly surpasses 50% with version 1.3.0 with a value of 60%. Version 1.4.0 increases the similarity in successive versions to a range from 72% to 75%. From version 1.5.3 to 1.6.0, it increases by another 10% and continuously aligns with 100% in version 1.6.7.

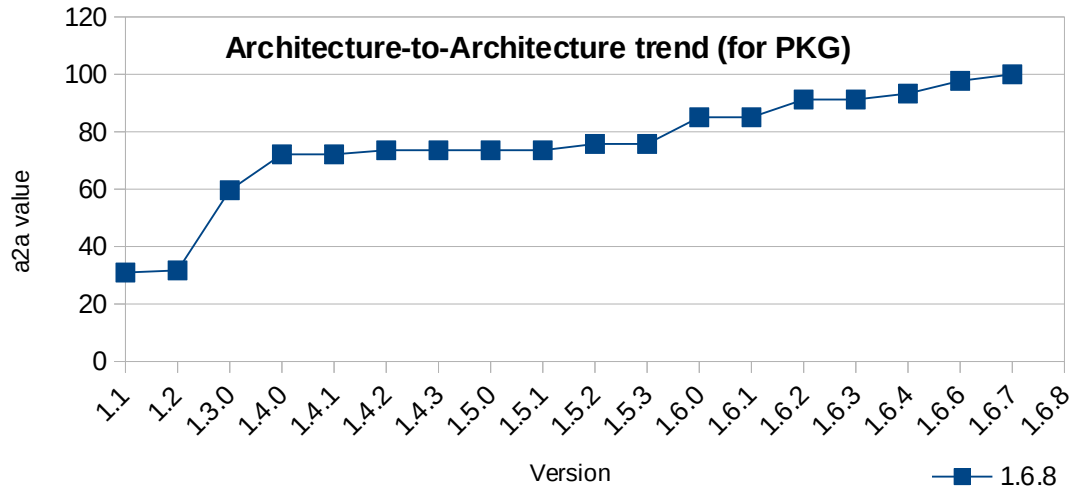

Figure 14: a2a trend for PKG recovery related to JackTrip version 1.6.8

Measurements of the cluster coverage for ACDC recovery, as shown in Figure 15, illustrate a similar evolutionary trend stretched to a starting value of approximately 0.2. On the one hand, the ascent from version 1.2 to 1.3.0 is less steep. On the other hand, there are steeper ascents between the other versions' transitions. However, a plateau around 0.54 and 0.6 with slighter increments in between from version 1.4.0 to 1.5.3 and full similarity with version 1.6.7 are identical characteristics compared to a2a metrics.

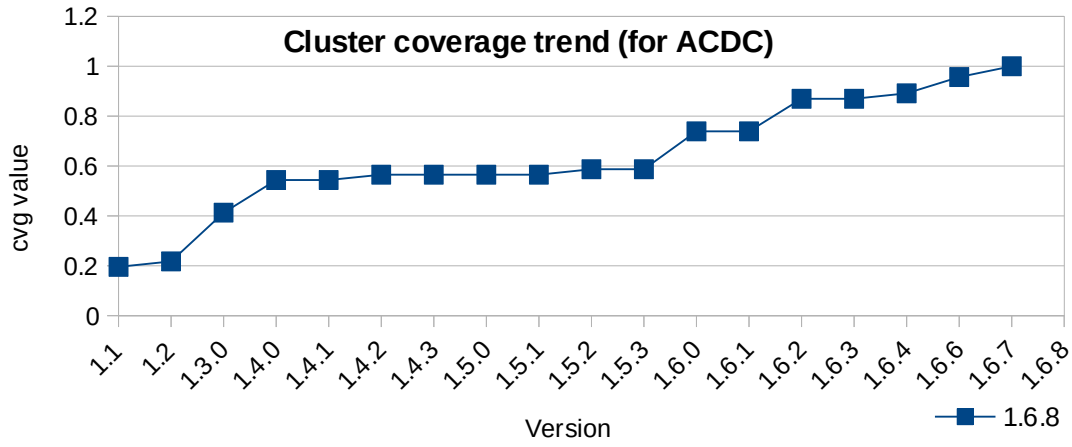

Figure 15: cvg trend for ACDC recovery in relation to JackTrip version 1.6.8

A situation of a maximum stretch in the trendline for PKG cluster coverage between the initial versions 1.1 and 1.6.8 is drawn in Figure 16. Starting with a value of 0, the cluster coverage increases in steps of one-third until full similarity is reached. The versions that are responsible for those steps in the evolution are 1.3.0, 1.6.0, and 1.6.2.

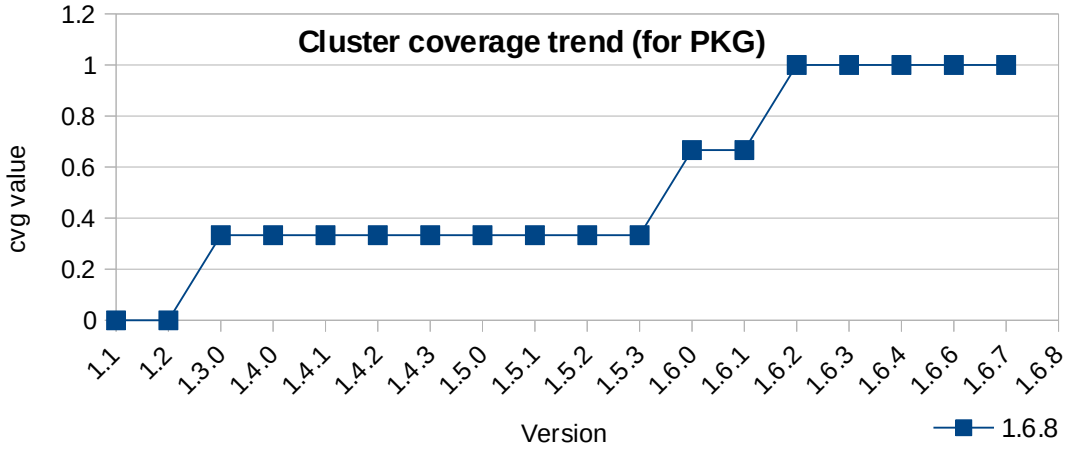

Figure 16: cvg trend for PKG recovery in relation to JackTrip version 1.6.8

Besides a2a and cvg values, measurements of architectural instability, inter and intra-connectivity and their extensions Basic- and TurboMQ all support in the illustration of the system evolution and identification of noticeable occurrences as described before.

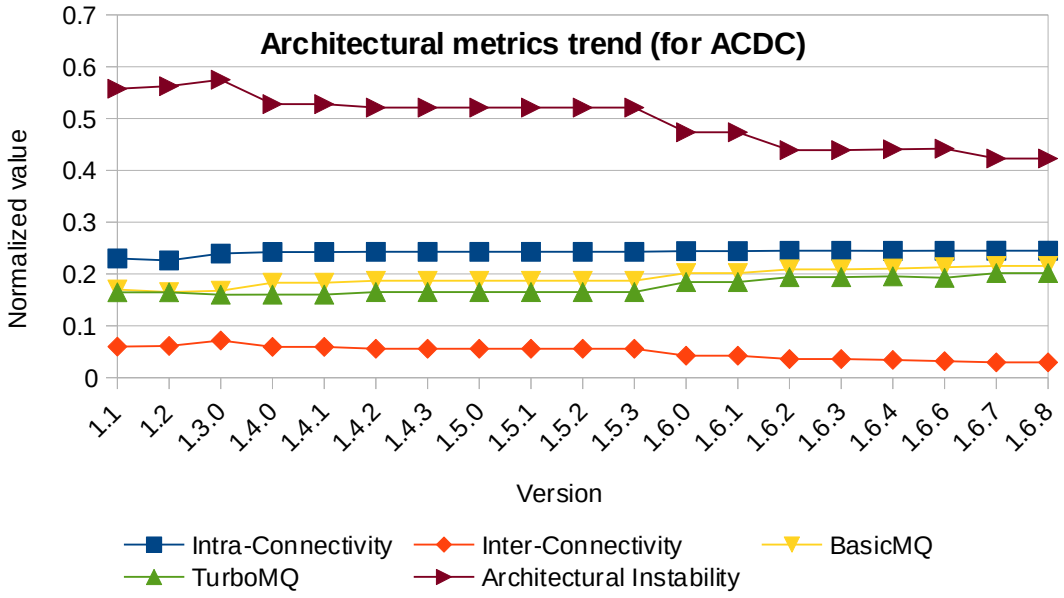

Figure 17: Trend of architectural metrics for ACDC recovery of JackTrip

In the ACDC recovered view as shown in Figure 17, the architectural instability decreases in several steps from values above 0.5 in the beginning towards 0.4 in version 1.6.8. First, it slightly increases from 0.5575 in version 1.1 to 0.5749 in version 1.3.0 and then decreases to 0.5278 in the next release. A slight decrease to 0.5212 occurs with the release of version 1.4.2 and persists until 1.5.3. With that release, the architectural instability decreases continuously in three steps to a final value below 0.4229 in version 1.6.8. One of those steps is another very slight increase from 0.4391 in version 1.6.3 to 0.4407 in version 1.6.4. The values of interconnectivity correlate with that trend, although they range from 0.0599 in version 1.1 down to 0.0295 in version 1.6.8. Conversely, the intraconnectivity shows anti-roportional behavior with increasing values in a tight range from approximately 0.23 to 0.25. Similarly, BasicMQ values increase in three steps during the evolution of JackTrip versions. In addition, the TurboMQ follows the antiproportional trend towards architectural instability with values ranging from 0.1602 to 0.1652 until version 1.6.0, and an increase up to 0.2018 is introduced in version 1.6.8.

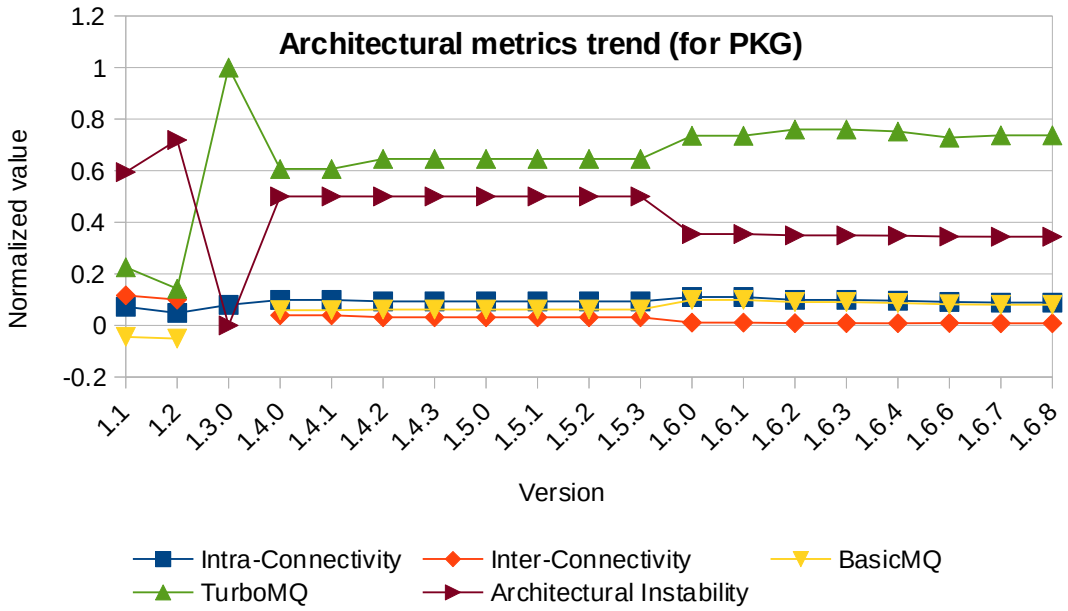

Figure 18: Trend of architectural metrics for PKG recovery of JackTrip

The metrics applied to the view of the PKG recovery depicted in Figure 18 show overall similar results as of version 1.4.0. The tendencies of decreasing architectural instability, interconnectivity and, in contrast, increasing intraconnectivity, Basic- and TurboMQ are

obvious, though absolute values are shifted to a higher or lower meanline, respectively. However, most suspiciously, in version 1.3.0, the architectural instability is exactly 0, while TurboMQ is exactly 1. The BasicMQ value for this version is missing and negative for versions 1.1 and 1.2. The former also applies to interconnectivity, which converged towards zero since version 1.6.0. Changes in version 1.3.0 seem to specifically distort the results of the PKG recovery method.

4.2.2 Trends of Architectural Smell Types

Counting the number of AS types per release indicates the trend of an overall increase in link overload with successive versions. The results are illustrated in Figures 19 and 20 for each recovery method.

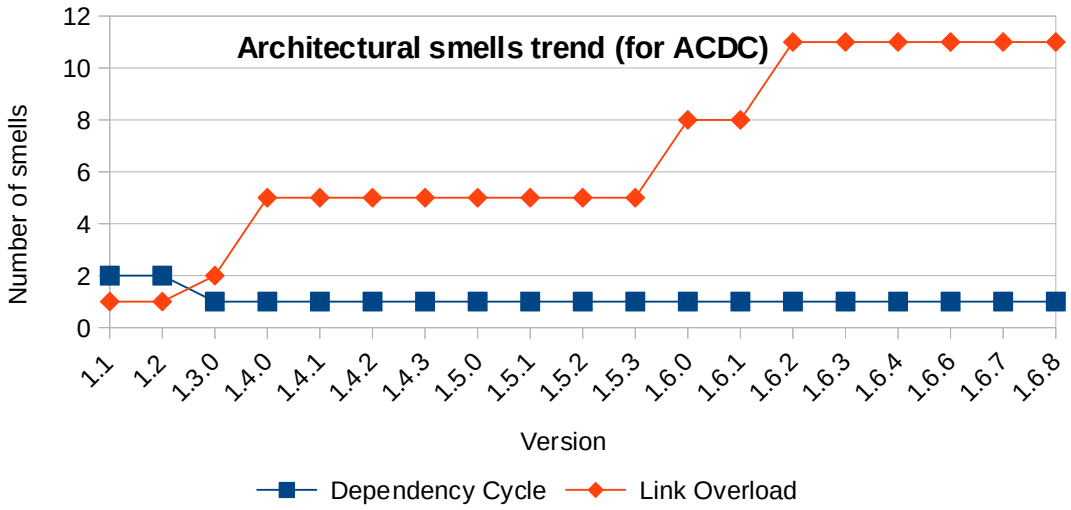

Figure 19: Trend of architectural smell types for ACDC recovery of JackTrip

Figure 19 shows the number of each AS type for the architectural view using the ACDC recovery method. Certain releases mark the introduction of new link overload smells with periods of unchanged presence. Version 1.1 started with one smell that remained in the next release. After another new smell in version 1.3.0, the number of smells increased to five with version 1.4.0 and remained stable until another increase to eight in version 1.6.0. Finally, version 1.6.2. increased the number of link overload smells to a total of 11, which has persisted in later releases. The number of dependency cycle smells also remained unchanged as of version 1.3.0. Starting with a total of two in version 1.1, the number has decreased to one with version 1.3.0.

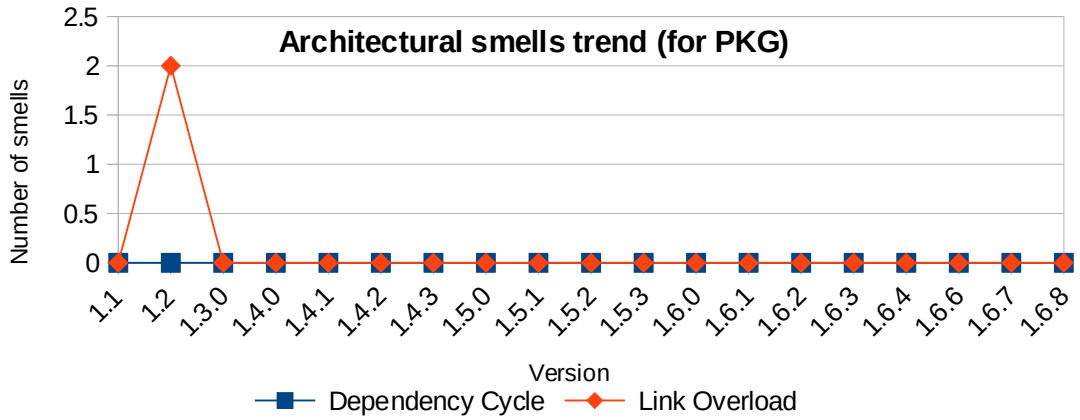

Figure 20: Trend of architectural smell types for PKG recovery of JackTrip

The trend of AS types for the PKG recovered view shown in Figure 20 experiences no smells of a dependency cycle type and a unique increase of link overload smells in version 1.2, which drops back to zero with the next release. Recalling the distorted architectural metrics results in version 1.3.0, this exception may be in relation to the design of the PKG recovery technique. Details on this are explained in Chapter 5 Discussion.

4.2.3 Visualization of Selected Architectures and Clusters

Noticeable differences in the metrics and AS types' counts require further investigation of specific versions and their clusters. In the following, visualizations of selected architectures and clusters are examined to illustrate prior results and find new anomalies for discussion. The representations use circles for clusters named by their filename, including the relative path, and lines with arrowheads pointing to dependencies.

Using visualizations of architectural views based on the PKG recovery, several cluster additions and removals can be observed in the evolution of JackTrip. Figure 21 shows that additional clusters were introduced with version 1.2 and completely eliminated in the next release. The cluster names, such as `external/includes/rtaudio-4.0.6`, indicate the directory structure of an external library that has been included in the project. In version 1.3.0, these subdirectories seem to have disappeared. It is not shown in the figure, but in version 1.4.0, a new cluster named `src/gui` was introduced, and yet another one named `src/dblsqd` was added with release 1.6.0. Until version 1.6.8, the three clusters `src`, `src/gui`, and `src/dblsqd` were persistent.

Figure 21: Visualization of the PKG architectures from JackTrip versions 1.1 to 1.3.0

The `src/gui` cluster, existing since version 1.4.0, experienced a constant growth of new dependencies in three steps until reaching a total of 33 in version 1.6.8. However, the total dependencies of the cluster `src/dblsqd` did not change.

Concerning this, visualizations based on the architectural view produced by ACDC recovery do not show accumulating dependencies for any file located inside the `src/gui` directory. The ACDC clusters that include `src/gui` in their filename have no more than one dependency, which is their own individual header file. This pattern seems to apply to other clusters throughout the evolution, as shown, for example, in `src/JackTrip-Worker.c` in Figure 22.

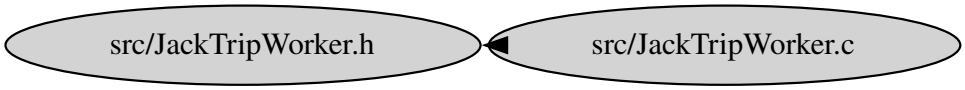

Figure 22: Header file dependency of an example ACDC recovered cluster

An exception is the cluster `src/jacktrip_globals.c` which had a high number of 12 dependencies in version 1.1; these increased to 13 in version 1.2 and decreased to one

pointing to its own header file as of version 1.3.0. The cluster `src/Settings.c` is another exception that seems to align its number of dependencies with version 1.3.0. Illustrated in Figure 23, the four dependencies prior to this version are `src/ThreadPoolTest.h`, `src/Settings.h`, `src/NetKS.h`, and `src/JackTripWorkerMessages.h`.

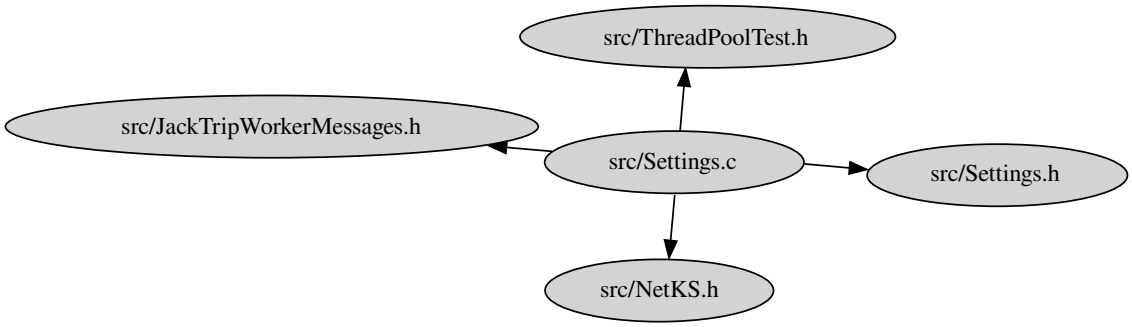

Figure 23: ACDC dependencies of the `src/Settings.c` cluster before JackTrip 1.3.0

Lastly, a visualization of the dependencies in the architectures of version 1.1 and 1.6.8 is provided in Figures 24 and 25, respectively.

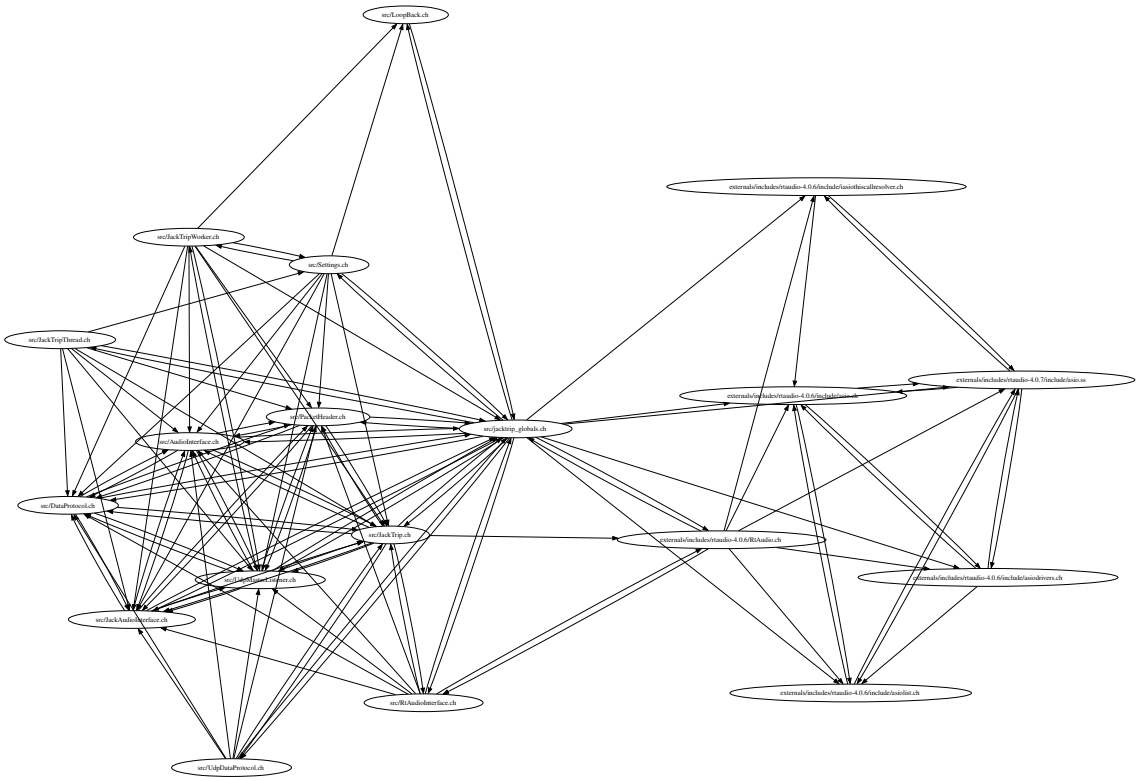

Figure 24: Visualization of the ACDC recovered architecture for JackTrip 1.1

In the evolution of JackTrip, a large number of new clusters were introduced. Comparing the clusters that are most frequently referenced, the `src/main.c` cluster was designated as a subsystem named `src/main.ss` by the ACDC recovery method as of version 1.4.0 and until version 1.6.8 (Figure 25). In earlier versions, this cluster was not available in the same form. Versions 1.1 and 1.2 comprise the cluster `src/jacktrip_globals.ch`, which is a target for most of the connections, as shown in Figure 24. Version 1.3.0 introduced a cluster named `src/jacktrip_main.ch` as a predecessor of the final `src/main.ss` subsystem cluster.

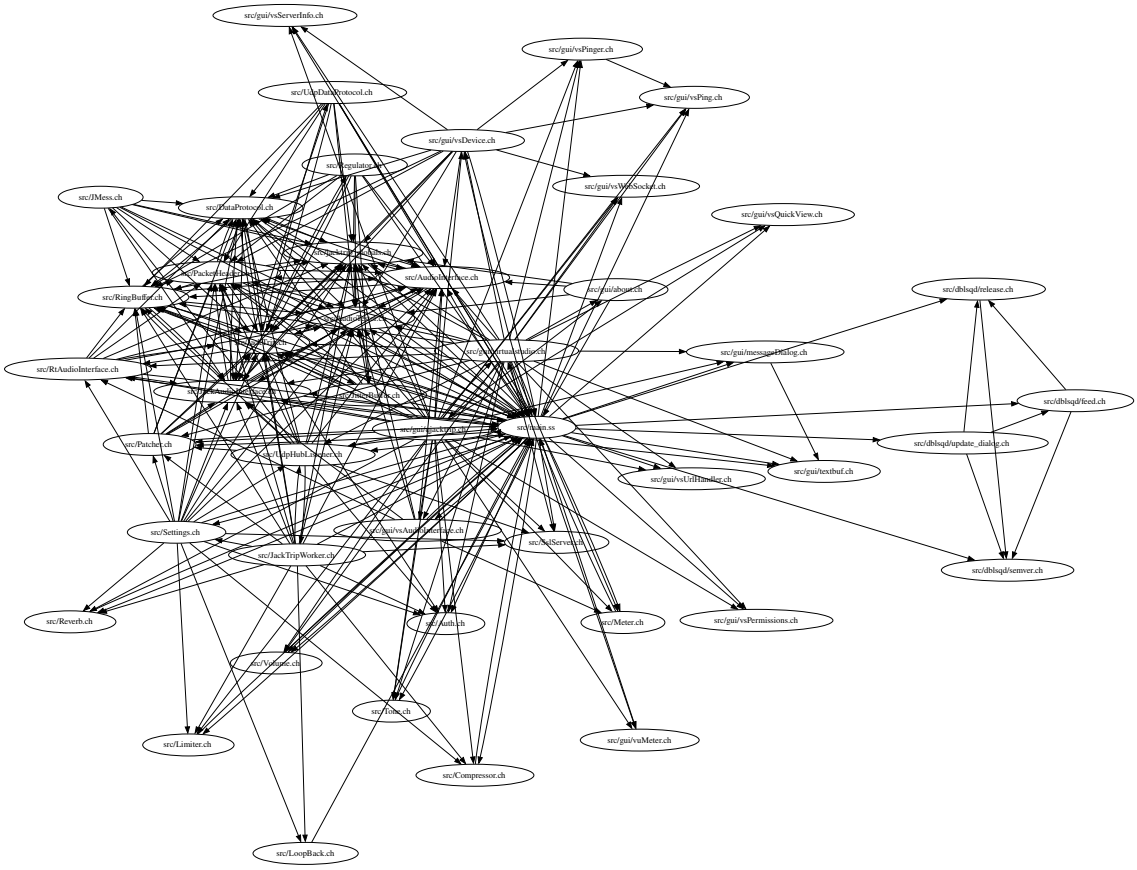

Figure 25: Visualization of the ACDC recovered architecture for JackTrip 1.6.8

Another salient feature is the overall low number of dependencies on and within clusters in the form of `/src/gui/vs*`. They first appeared in version 1.6.2. Before that, in version 1.6.0, a cluster called `src/gui/virtualstudio.ch` was introduced and seems to be the main dependency for the aforementioned clusters.

4.3 Thematic Analysis of Interviews

The findings in the analysis of the interviews reveal different perspectives on several aspects of the JackTrip project. There are controversies and agreements, such as the usage of third-party APIs or a desire to separate the core functionality into a library. Due to the nature of semi-structured interviews, the investigated topics reappear in the statements. In order to avoid duplicate quotations while focusing on different aspects of the research questions, the report is structured into the following main themes:

1. Evolution Speed,
2. Contribution Management,
3. Driving Factors for Development,
4. Motivations and Barriers for Engagement,
5. Codebase Perception,
6. Limitations of the JackTrip Protocol,
7. Problem Domain Peculiarities, and
8. Use Cases and Types of Contributors.

Each theme is described by the interviewees' individual perspectives using quotations as evidence. In the discussion (Chapter 5), the themes are directly related to other findings in order to answer the research questions.

4.3.1 Evolution Speed

In relation to a possible rushed evolution the interviewees were asked, if they have experienced a change in the speed of evolution compared to previous periods and how it affects their practice as developers. Changes in the speed of evolution were identified by all the interviewees, and most prevalent is the perception of a fast evolution during the COVID-19 pandemic in 2020. Interviewee 1 (IE1) distinguishes the overall evolution into several waves with active development cycles for achieving major milestones and times that are primarily characterized by the usage of the software. Table 4 lists the periods and summarizes some of the achievements to serve as a point of reference in the following. After initial research and academic usage as of 2000, the main development

took place from 2004 to 2009. Core technology, such as the hub mode, became stable and ready to use in 2010. Before the pandemic in 2020, the codebase seemed to be mature, and JackTrip had not gained many features while being used in academia. IE1 regards features added at that time as not relevant for core functionality.

Table 4: Development waves of JackTrip according to Interview #1

Period	Year	Description
1	2000 – 2004	Initial research and usage in academic environments
2	2004 – 2009	Main development and rewrites, e.g., adopting audio APIs and libraries
3	2010	Core technology (hub mode, etc.) becomes stable and ready to use
4	2011 – 2019	Mature codebase for academic applications such as teleconcerts
6	2020 – Present	GUI, commercialization and ease of use due to increased public interest

A rapid evolution in 2020 is described by Interviewee 3 (IE3). Although on a day-to-day basis in the commercial project he does not recognize the efficiency of development at first, retrospection reveals the amount of change:

> [T]hat kind of combination of the open source with what we're doing at JackTrip Labs, I think [...], is really evolving rapidly. [...] It was a pretty unique population base[,] and now it's a brand (Interview #3).

One of the urgent reasons for the speedup is increased interest and ease of use, which contributes to the former in the opinion of IE3. Another is the mutual relation between the OSS and the commercial project, described by IE1 as follows:

> [O]ften it's kind of a tandem thing between the open source JackTrip developments [...] [and] the commercial Virtual Studio stuff. And there'll be kind of [...] a back and forth. A new idea for the open source might immediately cause something to [be] release[d] in the Virtual Studio world[,] and vice versa. (Interview #1)

For Interviewee 2 (IE2), the pandemic was only the first change in the speed of evolution in 2020 as the commercialization by JackTrip Labs shifted the mode of development towards customer experience due to the integration of Virtual Studio and its implications on adding just another user interface.

> The second big change was when JackTrip Labs became bigger and contributed more to the JackTrip application. They have a website for Virtual Studios that you can create. And so[,] you don't have to host your own server. [...] But it is a big shift from developers that develop for

> themselves or for the parts of JackTrip they are interested in[...] to a mode of development where JackTrip is developed for customers that use JackTrip Labs Virtual Studio. And I mean, this is all in one application with the classic QJackTrip graphical user interface and the command[-]line interface. So[,] there are parts that are starting to clash. (Interview #2)

It appears that the number of contributions by JackTrip Labs affects IE2's practice as a developer in such a way that his time and knowledge are not sufficient to keep track of the changes in the codebase. One example is that creating a JackTrip connection via Virtual Studio is handled differently than in the classical GUI (QJackTrip) or the CLI.

4.3.2 Contribution Management

Another issue that has arisen due to JackTrip Labs' participation in the OSS is indicated by Interviewee 4 (IE4) with a focus on synchronizing different branches. Due to the hiring of full-time development staff and a huge number of commits, branches radically deviated, and code that was not quite ready for prime time led to a severe situation:

> It [...] came to a head with a release that really broke something bad there[,] [...] I think on one of the platforms [...]. Basically[,] they had to launch another release pretty much immediately after it[,] so it's ideally a situation that should never have happened, but I think that led to a lot of discussion about how we can make sure that doesn't happen again and what sort of processes we need to engage to make sure that the code gets into the code base in a timely fashion, but without breaking things. (Interview #4)

Before a clear delineation between JackTrip Labs and the OSS project had been defined, changes in the codebase seemed to affect both sides much more significantly. IE4 describes this period as “a time where people were just committing stuff really quickly to a point where it's like, well that's great that people are making these contributions. But suddenly now I have all this code here that is out of date and I need to refactor” (Interview #4). In the beginning of JackTrip Labs, IE4 worked on that side of the project while also contributing to the open source side, which complicated keeping track of changes in the former. A general strategy has been to regularly merge commits back into his own branch and quickly work around any clashes. This has also been an issue in periods with infrequent commits to the main development branch as contributors were used to maintaining their own separate branches, making an out-of-sync situation occur more often. However, the time between commits to the development branch decreased,

and pull requests happen more frequently today. Because of this, merges are more attractive, and even if manual fixes need to be done, they are fairly small according to IE4. In addition to that, IE3 explains that despite the small team size and the absence of a dedicated quality assurance organization, careful testing and the prioritization and inspection of bugs support building a resilient system in the long run:

> I think we try to test things as much as possible, but it's also normal that there's going to be a lot of things that slip through. [...] We review all the bugs continuously throughout the week[,] and we prioritize them. [...] [B]ugs always take priority over new development, if it's a certain criticality level or if it's impacting a certain number of people. We always bump that to the top and try to fix it right away. Sometimes we'll fix it […] [the] same day[;] just push it right out. So [I] think responding quickly is important, but also looking at how that bug [made] it in there, [and] how it happened (Interview #3).

IE3 refers to the process as a “mental postmortem,” tracking the origin of the bug and installing countermeasures such as checklists and automated tests to prevent bugs from slipping through. An example is the detection of improper package signing. They learned that this affected a few of the last releases, and they created an automated test check for signing the packages before releasing them. To IE3 bugs are an opportunity to not only fix the issues but also address their roots to avoid future occurrences.

In the history of the OSS project, IE1 relates to making mistakes due to frequent changes, primarily in the early periods before 2011. At the beginning, from 2000 to 2004, there was no community-based version control system such as Git. For that reason, no code reviewing or contribution happened, which limited the knowledge about bugs and the scope of fixing them. Only a small group of researchers—mostly one person—was working on the code at the same time. Rewrites from 2004 to 2009 incorporated new technologies and extended the codebase to gain new features. The first rewrite by Chris Chafe in 2004 integrated JACK to allow the mixing of two JackTrip instances running on the same computer in order to create a peer-to-peer mesh of three sites. This renewed version also introduced the Qt framework. Another rewrite brought hub-and-spoke functionality for connecting multiple clients to one central server and was created by Juan-Pablo Cáceres in 2010. As a result, later feature additions had to adapt to prior design decisions, as described in the following quote:

> [K]ludge upon kludge is like […] [when] you have something that was kind of adapted to make it work. And then you don't go back and straighten things out and make it beautiful. You go to the next feature. And then you adapt it some more. And you make it work, right? So[,] I wouldn't characterize JackTrip as having so much of that. But I do see that over time, as these rewrites happen, they [are] accommodating decisions that were made in the earlier version and having to adapt to them. Not that they were kludges, but that other decisions could have been made if things had been started from scratch. (Interview #1)

Consequently, the codebase grows, and structural aspects such as naming conventions or the way functions call other functions may remain unchanged in their original implementation. That long tradition of adding new features without subtracting earlier ones actually makes navigation in the code more difficult. The growth of the codebase is also connected to the motivation of developers. In the OSS project, voluntary contributions are mainly driven by passion, according to the perspective of IE1. Even though that activity might compete with their work, which is common in open source, new ideas inspire other people to engage and promote a great deal of code development. New ideas driving development can also originate from JackTrip Labs due to the shared codebase. For example, the introduction of GUIs resulted from what IE1 calls “when the ball gets rolling” (Interview #1) and certainly impacts the project management in his perspective.

4.3.3 Driving Factors for Development

One of the main drivers for the high development activity are upcoming demonstrations, shows, concerts, and conferences. The number of these events has increased since the pandemic through JackTrip Labs' promotion activities. However, as early as from the initial research, the opportunity for demonstrations at, for example, the Super Computer 2000 and 2001 convention, pushed development, as IE1 remembers. Due to the academic environment, external influences, such as people finishing research projects and completing PhDs, affected the pace of development. On the commercial side, shows such as the National Association of Music Manufacturers are pushing development, according to IE1. During the pandemic, the increased interest in projects for real-time online collaboration had a huge impact as meeting in person to play music together was not possible. Regarding this, IE3's motivation was to technically support a choir's rehearsals with 40 or even 200 members.

Apart from the previous explained motivational factors, IE3 also refers to time constraints and how they affect development in JackTrip Labs. The amount of work is high, and though more money could influence the development speed, time constraints persist. They are manageable by examining and reprioritizing tasks every day as it takes time to develop features; fix bugs; release new features; address issues arising from new features; and maintain and keep features up to date. Features are prioritized and chosen based on their impact. Those with a greater impact are usually implemented sooner.

This prioritization is shapes the project at all times in IE3's perspective as it is “the overriding factor for everything” (Interview #3). For this reason, time pressure is perceived even before development starts. The development of a feature must be scheduled carefully. A feature that is in progress and has bugs and customer support cases related to it creates time pressure. Moreover, the coordination of resources and deadlines of other developers are impacted by time pressure during the development cycle. In the opinion of IE3, such organization and execution of tasks always take longer than expected. However, the general workload remains practically unchanged. As long as there are not more people available, the workload is queued:

> I'd say that the workload that's queued is changing. You may have [a] workload that changes to the most critical priority workload that's not being worked on. Let's say to me that changes depending on where you're at and how busy everybody is, but I think the total amount of productivity is fairly consistent as long as you're optimizing that productivity. (Interview #3)

Management of the workload is strongly supported by the used process model. IE3 states that JackTrip Labs follows the Scrum[41] model using two-week sprints and allowing adjustments during iteration. Necessary adjustments are taken care of in check-ins as exceptional cases, for example a critical issue. Regular sprint planning meetings and sprint reviews serve as foundational organization. Holidays and vacations of developers can be taken into account, and the amount of work for each task is assessed. Typically, this does not work out from the beginning as every team needs about three months to move along, and the clock resets if the team is changed. For that reason, planning beyond two weeks is overly optimistic in IE3's opinion as prioritization of the backlog items is constantly changing. He explains it as follows:

41 https://www.scrum.org

> If they're not changing, then you're probably doing something wrong. Looking at the next two weeks and every two weeks is really the way to stay optimized[,] and I think that works pretty well. It's hard for people to get into that mindset sometimes[,] [e]specially if they come from an organization that's more organized on a [...] waterfall model. (Interview #3)

For the OSS project, there are no planning techniques according to the interviewees. However, IE2 characterizes the community culture and primary focus of the development before JackTrip Labs was founded in relation to the on/offboarding process.

4.3.4 Motivations and Barriers for Engagement

When IE2 started committing to JackTrip, he described other contributors as engaged in development for their own use, in their spare time, or in an academic environment, but not for customers. His participation became comparable to a hobby in that he did maintenance and cleanup. Another contributor joined to perform a gread deal CI work and create GitHub actions to check whether JackTrip builds on Windows and mac OS in addition to Linux. It has also been the case for developers to work on one feature and then leave the project, which IE2 classifies as easy to do in an open source project.

Other interviewees related similar reasons for engagement in the project. They typically started with smaller bug fixes, feature additions or maintenance tasks and became involved over time. From a usability perspective, the lack of a graphical user interface at the beginning of the pandemic might have been an obstacle to increasing interest in JackTrip. Therefore, IE4 started working on QJackTrip, which has been integrated since he joined the project. His motivation was characterized as follows:

> I had a lot of staff and students at the university who were really interested in the project and interested in using it until you started talking about the command line [...]. So that kind of became my main focus then. (Interview #4)

IE3's critical requirements are low latency and scalability as the intention is to support a high number of simultaneous users. By testing and comparing it to other solutions, IE3 found that JackTrip has great latency characteristics and is able to scale. He refers to it as a “live monitor” of his network as the measured latency of less than a millisecond reflects the latency of his test network. In terms of scalability, the multithreading architecture and hub-and-spoke operation mode are other advantages stated by IE3:

> I found JackTrip [to be] the only thing that could scale. Everything else either wasn't designed to scale from a multithreading architecture perspective or it just could[n't] scale like peer-to-peer[-]based approaches because they hit other walls very quickly. (Interview #3)

In relation to such design decisions, several aspects of JackTrip were elaborated on the interviews. Most noticeable, all the interviewees described the notion, desire, or outlook to transform, rewrite, or reinvent JackTrip as a library that implements the core technology or core components to ease its reuse and separate application-specific code such as operation modes and user interfaces. An application programming interface (API) would allow that library to be integrated by other projects that may use different frameworks and technologies, in contrast to the current state of a monolithic standalone application heavily relying on Qt, according to IE3.

> I think, if it started today, it would use [...] modern C++ libraries, C++14[,] and a lot less of the Qt stuff. I think it would have a cleaner distinction between [...] the library that could be reused by different implementations of JackTrip and the various end applications. (Interview #3)

According to IE3, the adoption of Qt features has been primitive in relation to current capabilities. Since 2000, features such as threading and signals and slots[42] have become part of the newer C++ standards and can be used with a more modern compiler. Especially for nongraphical programming, using Boost[43] appears more natural to him. IE2 also relates to the way of adopting Qt and modern C++ standards regarding issues arising after the introduction of graphical user interfaces:

> Although JackTrip […] was even written in Qt before it got a graphical user interface[...] [s]o JackTrip at the beginning was just a command[-]line interface. I think a lot of the problems with managing the codebase in JackTrip started [...] with the introduction of graphical user interfaces. (Interview #2)

He identifies a disadvantage for the management of settings when introducing GUIs. At runtime, the command-line flags and arguments are parsed by the settings class for creating the JackTrip instance instead of the other way around. Apart from the confusion that is raised by such a peculiarity, in IE'2 opinion, it is irrelevant for a CLI application:

42 https://doc.qt.io/qt-5/signalsandslots.html
43 https://www.boost.org

> But when you have a user interface where you can change settings [...][,] this approach doesn't work. […] [T]here are actually two different graphical user interfaces, one for the Virtual Studio, and [...] QJackTrip. And QJackTrip was integrated into JackTrip. So[,] because of this approach, you have the management of your settings at two different places in the JackTrip codebase. (Interview #2)

4.3.5 Codebase Perception

Referring to the structure of the codebase, IE2 most significantly emphasizes barriers to becoming familiar with it. Difficulties in placing features in the right place and the intertwining of OS macros in several parts of the code, in addition to the twice-settings management, are some examples that are still being addressed. Conversely, IE4 estimates the barriers to engagement as similar to those in other OSS projects. Though he acknowledges an increased codebase due to JackTrip Labs contributions, it is not relevant for implementing features in the OSS, according to his explanation:

> [I]f you're just planning on looking specifically at the open source side of things, there's less to get your head around. But you still have to work out which parts of the code are important to what you want to do with the project. So I guess now that [there]'s a bigger code base[,] maybe one of the barriers to entry is just getting accustomed to the code. But also[,] it's the same sort of thing that exists for all sorts of projects where there's going to be an initial learning curve no matter what you get involved with[,] and it's always going to take time to sort of come to grips with those things. (Interview #4)

In more detail, IE2 explicitly relates the difficulties of maintaining the code to the cross-platform approach. In his opinion, the common parts of a software are supposed to be a library that is shared by single CLI, GUI or special Virtual Studio clients. This would allow employing different frameworks for Windows, mac OS and Linux. A path to this solution, as explained by IE1, might be another rewrite called HackTrip[44]. It is a very efficient, more modern version of the JackTrip hub client written from scratch. Instead of JACK it uses RtAudio as the audio backend and Qt 6 as graphical framework. IE1's motivation has been to reimplement the basic operation of JackTrip and start with a GUI to create an application he fully understands. Though it is intended for his own purposes, IE3 considers it an example foundation for refactoring:

44 https://github.com/cchafe/hacktrip

> I'd see the next step of that as trying to take out as much of the Qt stuff that's already part of a modern C++ standard. To me[,] that would be a great next pass on that. And then probably rolling that refactor back into the larger project, re-[imagining] what [...] the server look[s] like, if it's built on top of HackTrip, for example. (Interview #3)

The aspect of adopting a modern C++ standard and dropping Qt reoccurs regularly in the statements of IE3. In his answer to the question of whether cohesion and coupling are considered when discussing parts of the system, he explains that in the commercial Virtual Studio project, there has been the opportunity of building modern microservices and follow best practices and principles while starting from scratch. In contrast,

> On the open source side[,] I don't know it as well as other people. I've haven't [made] a bunch of changes in there, but I know there's a lot of coordination across classes using signals and slots. In some ways[,] I think it's good[;] in some ways[,] I think it's unnecessary that things are somewhat coupled together through signals that don't necessarily need to be like or shouldn't be as coupled as they are. And I think it makes it hard to understand how those things work together in some senses. (Interview #3)

In IE4's perception cohesion and coupling have were considered in the development meetings he attended. Discussions about architectural choices took place and mostly worked out in the long run. In comparison to other projects, he refers to how conversations were “a bit formal” (Interview #4) but appreciated that. Due to time constraints he has not attended those meetings recently and thus can not testify to current trends.

4.3.6 Limitations of the JackTrip Protocol

Regarding JackTrip as a protocol, IE2 emphasizes that there is no path to extend it due to the lack of a version indicator in the protocol header or means for checking the supported features of a peer. This complicates adding, for example, OSC or MIDI channels, as shown in the following quote:

> That's a part I really want to change, to check where the boundaries of the JackTrip protocol are right now and how we could create a new JackTrip protocol that is versioned, that would be capable of having different channel types like OSC and MIDI and maybe even video and prioritize between those different channels[…] [b]ecause the most important part is still the audio. And if you send additional data that has the same priority, then you could [have] problems with the audio signal. So that's the reason I want to have this all[-]in[-]one protocol. (Interview #2)

Issues that constantly arise from the limitations of the protocol are also elaborated on by other interviewees. IE4 agrees with the difficulty of adding extensions and new features and mentions the necessity of using tricky workarounds while not breaking backwards compatibility. He tracks the origin of these conditions back to challenges with the available technology at the time of JackTrip's early development. Design decisions that were made back in those days accounted for other requirements and less capable devices. As an example of how this affected further development, he describes the implementation of sending an exit code to indicate the intentional termination of a connection to the server to distinguish it from a dropped connection. One of the challenges has been a tiny protocol header and small packet size resulting from a low frames-per-period setting in JACK. From the perspective of a fast and responsive transfer in earlier times of the Internet it is preferable to decrease the amount of additional information for each audio packet, but this also complicates adding, for example, control commands into the protocol. This condition has required creative ways of circumvention. It has also become an obstacle in the course of implementing an authentication mechanism into the protocol as the initial TCP handshake only sent a 16-bit port number. IE4 exploited the fact that a 32-bit integer field is used. Therefore, hijacking the 16 unused bits does not break backwards compatibility and maintains the protocol. However, those bottlenecks urge the need for a new protocol version:

> But I think there [are] always various talks at developer meetings about how we can get around those things[,] and who knows, eventually there may be [...] a JackTrip 2.0. We just [decided] to re-engineer things, but I think that would be a big project[,] and at the moment[,] I think everyone else is focused on other things. (Interview #4)

Another example of an issue that occurred in relation to the way the protocol was designed is the IPv4 socket-sharing between two parts of the program. To IE4, it appeared to be an ingenious workaround out of the socket specification, but for that reason, it did not work with IPv6 and needed to be replaced. Additionally, large chunks of original code were renewed as a means of transforming the system to work asynchronously, which solved many issues and improved maintainability while preserving the same protocol behavior. However, addressing limitations in the protocol itself still requires creative solutions or breaking backwards compatibility from IE4's perspective:

> But once again[,] I think that's the sort of thing where we would need to get a lot of the stakeholders in the room together to talk about what's the best way to define an address space to do things here. [...] I think we haven't sort of made any real progress on that at the moment. (Interview #4)

The quote indicates that there are solutions being discussed, but have not been realized so far. For IE3 the desire to perform an overhaul might also be connected to the hesitation to create technical documentation. He acknowledges that this is a barrier in itself and also identifies it as a barrier for onboarding in the project:

> [L]ooking at [...] my experience in terms of getting on board with it[…] [and] also my experience bringing other engineers into that project[,] I'd say one of the barriers is [a] lack of technical documentation. [...] [T]his is the whole system[;] these are the different pieces of it, diagramming it out. “Here's how it works.” I don't think that exists, yet[;] probably should. (Interview #3)

4.3.7 Problem Domain Peculiarities

Regarding the specifics of the problem domain, IE1 points out that hub mode, as a server that aggregates and mixes the audio streams to send them back to client spokes, is a concept that is less common for other applications such as video streaming. Bandwidth and CPU requirements increase with the number of participants in a session, and the system must remain capable to guarantee real-time behavior. The configuration of geolocated Virtual Studio JackTrip hub rooms reflects how this is taken into account.

IE4 supports the perspective that the problem domain had relevance to the initial decisions in development and hints at details on how the project leveraged other existing software tools. JACK, for example, has been almost a necessity in terms of achieving a low audio latency locally. As the latency of the Internet connection cannot be minimized to a greater extent, it is logical to approach the problem at that point. That requirement seemed to govern a lot of decisions, as IE4 outlines:

> [I]nitial decisions about the project were [heavily based] around minimizing latency at the cost of everything else[,] and that's where some of those other bottlenecks [...] have come into play. And now there's a bit more of a focus on things being user friendly that sometimes clash with the low latency ideals, but I think there's been a lot of work to kind of find the best compromise between the two as much as possible. (Interview #4)

Today, relying on JACK as a third party becomes a liability as the latency requirement can be slightly neglected, especially on platforms like Windows, as there are many conflicting audio APIs, legacy APIs, and peculiarities with the sound system. In IE4's opinion, from the JackTrip Labs side, there is a push to get away from the dependency in order to allow less technically inclined users better access. In exchange for slightly lower efficiency, the project might probably gain more flexibility:

> So SonoBus, for example, was able to leverage things like the JUICE toolkit. [...] JackTrip really [was] kind of created largely from scratch. The main frameworks that it uses are just the Qt frameworks, which of course are not particularly geared towards audio or anything like that. There's no ready-made audio toolkit that it was using other than[,] of course JACK [...] whereas [...] SonoBus has just had ready access to that sort of flexibility right from the get-go simply because of the era in which it was coded. (Interview #4)

Adapting the requirements to gain flexibility applies not only to the audio API, but also to the characteristics of Internet connections nowadays. However, that sort of high level of efficiency for transmitting audio data does not matter any more, according to IE4.

4.3.8 Use Cases and Types of Contributors

To ease the usage of JackTrip, IE3 explains that the user experience is considered from the perspective of a musician and geared towards his or her expectations. By watching and adapting to trends in the music technology domain, the concepts of popular DAWs, such as the look and feel of volume meters and sliders, are taken into account. IE2 adds that JackTrip comes from an artistic background and has actually been developed by musicians to create an experience similar to playing together in one room. However, there are capabilities that only apply to certain use cases:

> For example, JackTrip is capable of transporting ambisonic signals. If you have third[-]order ambisonics, like a spatialization system, you can use this 16-channel signal with JackTrip [...]. But that's not the common musician[s] that [want] to play with their band over the Internet. (Interview #2)

The ambisonics feature is an example of specifics that contributing users of JackTrip might have become accustomed to. Concerning other use cases, IE4 relates to his own academic purposes and his practice as a musician and ensemble member. He recognizes that the commercial Virtual Studio service takes away the burden of running his own

servers and acquiring the details of networking knowledge such as port forwardings. However, in an academic setting and with institutional resources where students can create their own point-to-point networks, there is no need to depend on a third-party infrastructure. Musicians who used JackTrip before are comfortable with the prerequisites for setup as they have already invested in their own infrastructure and equipment without relying on paid servers.

In the line with academic purposes, IE1 elaborates on his usage in researching the topic of wide area Internet reverberation. As the original intention was to use the network delay with a plucked string physical model for recirculation, newer features, such as the hub mode and multi-channel capabilities, allowed further research on the topic with a different focus before the pandemic period:

> The idea here was [that] if you have a bunch of sites connected to a hub, pretend each of those connections is a reverberator itself, like a room. It'd have to be multichannel because it has to have different reflections in the reverberation. And you combine several sites. It's sort of like having a palace with different rooms. And you can open and close the doors. And you can hear the acoustical effect of playing in that room. (Interview #1)

For IE1, this created an opportunity to implement the HackTrip rewrite, and he explains details for setting up a connection. He generally indicates that the practice of an engineer and musician is necessary to approach the problem domain as it is important to evaluate results from the perspective of a musician and draw suggestions from experiments:

> [O]ften[,] I've made the mistake myself of saying, well, we're going to have a population of musicians, [and] we're going to have a population of engineers, and they'll talk to each other. But that's a mistake because a lot of the engineers are musicians, and a lot of the musicians become engineers very quickly. And it's particularly important to have good ears and be able to talk about what you're hearing. And so[,] everybody does naturally have that capability, and they develop it[…] [w]hether they're musicians learning to network dropouts or engineers listening to audio quality. (Interview #1)

Coincidentally, the contributing developers are characterized as what IE1 would call “computer musician[s];” (Interview #1) they appreciate audio and audio quality by using their ears rather than approaching JackTrip development and the problem domain of low latency audio streaming for networked music performance in an abstract way.

5 Discussion

In this chapter, the results are discussed with respect to the theoretical background. The subchapter titles represent the main findings and relate different sources of evidence to the theories and backgrounds in chapter 2. The structure is oriented toward the order of symptoms for AE as described in subchapter 2.3.4. However, due to the intertwining, they tend to reoccur. The discussion begins with the architectural metrics interpretation.

5.1 The Impact of JackTrip Labs

The metrics described in subchapter 4.2 show several indicators of architectural change in the evolution of JackTrip. First, the trends generally tend towards the stability of the project and are consistent with observations of other systems also evaluated with ARCADE Core, according to its current maintainer. In its early development (versions 1.1 to 1.4.0), JackTrip seemed to be significantly more unstable due to the fact that the a2a and cvg values changed dramatically, as illustrated in Figures 13 to 16. The architectural instability first significantly decreased in version 1.4.0 according to the ACDC recovered view shown in Figure 17. Most puzzling is that it becomes even more stable between versions 1.5.3 and 1.6.0. The step to a new minor version usually shows a larger break in the architecture than do patch versions. As the a2a metric can only measure the similarity of clusters as a composition of the system and not the interactions between or within those clusters, or the entities that make up the clusters, it only shows to what extent the whole system remain stable. If parts of the system have changed, they are not detected by this metric.

A reason for that stabilization might be related to JackTrip Labs' maintenance in the codebase for integrating Virtual Studio, which can be derived from the changelog item “(updated) code cleanup and maintenance” in Table 3. Another aspect is that for the commercial Virtual Studio project, JackTrip Labs has been able to follow best practices and start with a microservice architectural style resulting in higher cohesion and lower coupling between components in that part of the codebase, thus contributing to an overall decrease in instability while more clusters are introduced. Though IE2 perceives clashes in the codebase since JackTrip Labs began collaborating, architectural metrics

for ACDC recovery suggest an overall trend towards stability despite shorter release periods, as illustrated in Figure 12, would indicate a rushed evolution instead. The development speed is characterized as rapid by IE3. Another interviewee used the word “tandem” to describe the cooperation between the commercial project and the OSS.

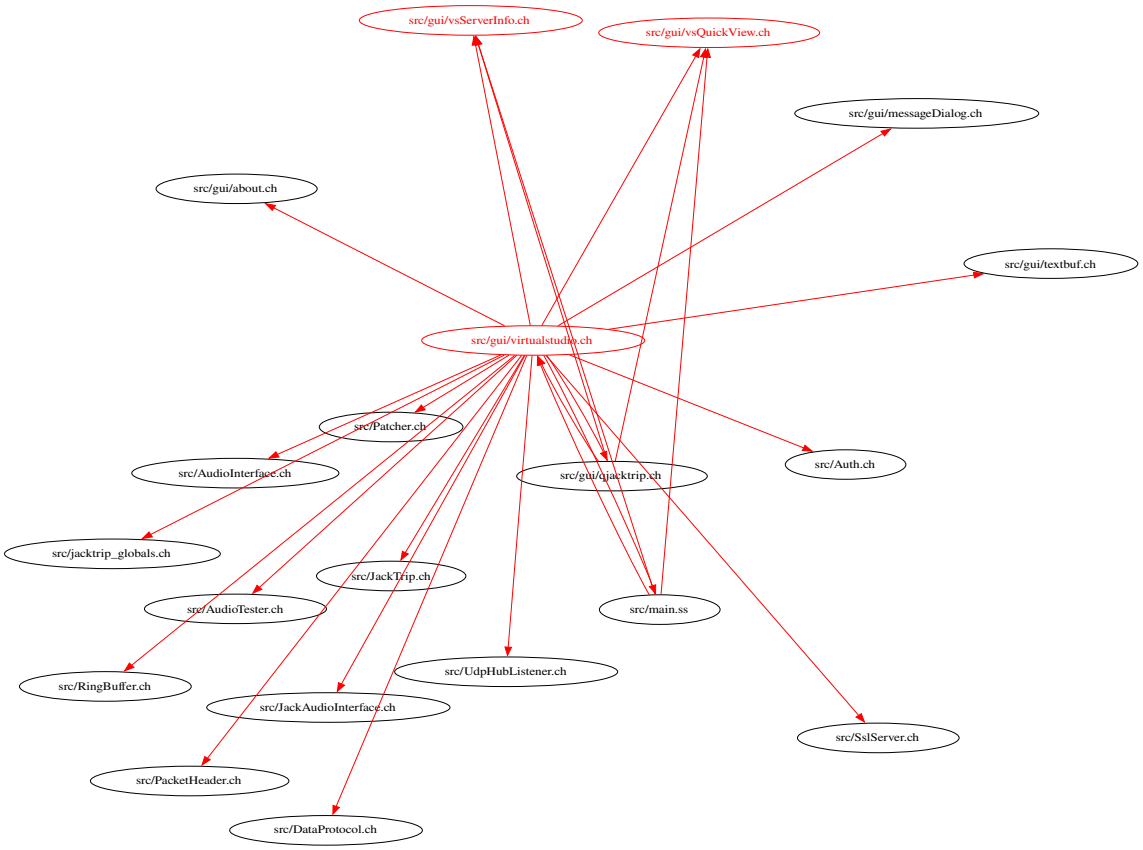

Figure 26: Virtual Studio dependencies of JackTrip 1.6.0 from ACDC recovery

The visualization in Figure 26 is a modified ACDC recovery view containing only Virtual Studio clusters and their direct dependencies. It shows `src/gui/virtualstudio.ch`, `src/gui/vsServerView.ch`, and `src/gui/vsQuickView.ch` with their connections to or from other components in red. These other clusters use a black color, and their dependencies to or from other components are removed a more detailed view. Obviously, Virtual Studio dependencies are fairly manageable as they do not spread across the codebase in version 1.6.0. However, that progressively changes until version 1.6.8, visualized in Figure 25, and is also indicated by a significant increase in architectural smells as of version 1.6.0, which is a typical symptom for AE caused by a rushed evolution.

5.2 Evaluation of a Rushed Evolution Symptom

As described in subchapter 4.2.2, the number of link overload smells increases from five to eight between the versions 1.5.3 and 1.6.0. Further smells occur after the release of 1.6.2, resulting in a total of 11 persisting until version 1.6.8, as shown in Figure 19. Though this might seem counterintuitive compared to the trends of other architectural metrics, a closer look at the definition of architectural smells supports a meaningful interpretation of the results. First, their occurrence can mark times in the evolution when architectural problems are introduced or removed. The ramifications of these problems must be identified to provide sufficient information about an actual issue in the codebase. For example, an AS can indicate a dependency cycle that never actually becomes an issue. Treating a smell has a very good chance of resolving the issue, and it indicates the likelihood of an issue occurring eventually.

Table 5: Involved clusters in smells of JackTrip 1.2 for ACDC recovery

ID	Smell type	Involved clusters
1	Dependency cycle	externals/includes/rtaudio-4.0.6/include/iasiothiscallresolver.ch externals/includes/rtaudio-4.0.6/include/asiolist.ch externals/includes/rtaudio-4.0.6/include/asio.ch externals/includes/rtaudio-4.0.6/include/asiodrivers.ch
2	Dependency cycle	src/UdpMasterListener.ch src/RtAudioInterface.ch src/JackTripThread.ch src/UdpDataProtocol.ch src/JackTripWorker.ch src/JMess.ch src/JackAudioInterface.ch externals/includes/rtaudio-4.0.6/RtAudio.ch src/LoopBack.ch src/DataProtocol.ch src/Settings.ch src/PacketHeader.ch src/JackTrip.ch src/jacktrip_globals.ch src/AudioInterface.ch
3	Link overload	src/jacktrip_globals.ch

Secondly, the detection of smells depends on the recovered architectural view and its limitations. Regarding this, the reduction of the dependency cycle smells in the transition from version 1.2 to 1.3.0 as shown in Figure 20 is another example. Tables 5 and 6 list the involved clusters for those versions. Comparing the involved clusters suggests

that the directory structure must have significantly changed. The metrics for both recovery techniques use string comparison to determine whether an entity is new to a version. If the entity changes name or location, the metric regards it as different from the previous version's. While ACDC also considers other information and uses the graph dominator pattern to create clusters based on the dependencies between entities, PKG's algorithm only uses the directory structure to create the architectural view, which leads to several distortions in the results.

Table 6: Involved clusters in smells of JackTrip 1.3.0 for ACDC recovery

ID	Smell type	Involved clusters
1	Dependency cycle	src/AudioTester.ch src/RtAudioInterface.ch src/Reverb.ch src/JackTripThread.ch src/UdpDataProtocol.ch src/JackTripWorker.ch src/JMess.ch src/JackAudioInterface.ch src/DataProtocol.ch src/LoopBack.ch src/Settings.ch src/RingBuffer.ch src/PacketHeader.ch src/UdpHubListener.ch src/Compressor.ch src/JackTrip.ch src/jacktrip_main.ss src/JitterBuffer.ch src/Limiter.ch src/jacktrip_globals.ch src/AudioInterface.ch
2	Link overload	src/JackTripWorker.ch
3	Link overload	src/jacktrip_main.ss

Examples are the exception of two link overload smells occurring in JackTrip release 1.3.0, as shown in Figure 20, and noticeable differences in the visualizations of the architecture from version 1.1 to 1.3.0 (Figure 21). This is related to the addition of multiple subdirectories inside `externals/includes` as a result of updating RtAudio to version 4.1.1 and including its directory structure, which was resolved in the next release. Another distortion caused by directory structure changes results from the missing and incorrectly calculated metrics in Figure 18. This is based on the fact that the value for TurboMQ is only exceptionally high under two circumstances: either the majority of the

components contain only up to three entities, or the architecture contains only up to three clusters, which is true for JackTrip in the PKG recovered view since version 1.6.0, as described in subchapter 4.2.3. If there was no design flaw, these values would normally suggest a fantastic level of modularization. In fact, it is an indicator of an exceptionally low number of modules. The extreme divergence between the TurboMQ results for ACDC and PKG suggests that JackTrip could significantly benefit from a restructuring of its directory structures to better represent the internal structure of the code itself, while the code itself is well modularized, that is the system is well structured in its dependencies, as the results from ACDC indicate the structural complexity of the system.

ARCADE Core's author notes that the results of both inter and intraconnectivity, and TurboMQ, as defined by Mitchell, experience a skew to the left due to a design flaw. Therefore, typical results range from 0.05 to 0.2 and hardly ever go over 0.33, instead of using the full range from 0 to 1. Taking that into account, JackTrip's connectivity, as illustrated in Figure 17 for the ACDC recovered view, has remained mostly stable throughout its evolution. Cohesion slightly increases from the beginning until version 1.3.0, and coupling decreases over the course of its evolution. This is a good indicator of encapsulation, suggesting that components make better use of their own resources while not relying on too many external resources, as depicted by Figure 22 and described in subchapter 4.2.3.

5.3 Governance of Design Decisions

An exception to the aforementioned highly cohesive structure is the `src/Settings.c` cluster in versions 1.1 and 1.2 (Figure 23). This implementation file has four dependencies on other header files. IE2 relates this class to issues arising when GUIs have been introduced due to the fact that settings need to be managed at two different places in the codebase. Referring to Tables 5 and 6, that cluster is part of the one persistent dependency cycle smell shown in Figure 19 that continuously affects an increasing number of clusters in the course of the system evolution. In version 1.1, the dependency cycle contained 14 clusters that are persisted in version 1.2, adding `src/JMess.ch`, which slightly increased the total number to 15. The next release adds six more clusters, and this trend led to a total of 30 clusters in version 1.6.8, as listed in Table 7. Though the number of

dependency cycle smells decreased in the course of JackTrip's evolution, one and the same smell affects an increasing number of clusters as of version 1.1, and the `src/Settings.c` cluster has been part of it since the beginning. Additionally, other clusters seem to persist in newer releases once they become part of that dependency cycle.

Table 7: ACDC clusters involved in the dependency cycle in JackTrip 1.6.8

ID	Clusters involved in the dependency cycle
1	src/RtAudioInterface.ch
2	src/gui/about.ch
3	src/UdpDataProtocol.ch
4	src/main.ss
5	src/DataProtocol.ch
6	src/Settings.ch
7	src/PacketHeader.ch
8	src/Compressor.ch
9	src/gui/vsDevice.ch
10	src/gui/vsAudioInterface.ch
11	src/JackTrip.ch
12	src/Volume.ch
13	src/gui/virtualstudio.ch
14	src/gui/qjacktrip.ch
15	src/Limiter.ch
16	src/jacktrip_globals.ch
17	src/Regulator.ch
18	src/AudioTester.ch
19	src/Reverb.ch
20	src/Tone.ch
21	src/JackTripWorker.ch
22	src/JMess.ch
23	src/JackAudioInterface.ch
24	src/LoopBack.ch
25	src/RingBuffer.ch
26	src/UdpHubListener.ch
27	src/Patcher.ch
28	src/Meter.ch
29	src/JitterBuffer.ch
30	src/AudioInterface.ch

At this point, it is obvious to ask to which extent and how SA is governed in the broader project. In subchapter 4.3.5, IE4 mentions that considerations of coupling and cohesion are part of regular development meetings and that architectural choices mostly worked out in the long run. For the commercial Virtual Studio project, IE3 relates to being able to use modern microservices while starting from scratch. The OSS project relies on Qt's signals-and-slots feature instead. In his opinion, most of the Qt features used in the codebase are already part of a modern C++ standard, making them superfluous as a dependency.

The adoption of the Qt library in the course of the project is an architecture decision that may have been eventually superseded by another one suggesting to adoption of a newer C++ standard for using built-in features in order to make the codebase more independent of external libraries. Without tooling and a process to document, manage, and communicate such decisions (e.g., by using ADRs as described in subchapter 2.2.4) their status, necessity, and implications are not transparent in the project. As there is no systematic way of making decisions, this leads to possible accumulation and loss of information, which lead to unconsciously choosing suboptimal solutions. An architect would be responsible for ensuring compliance with an architecture decision or suggest and justify modifications to it due to changing environmental conditions. ADRs would be able to serve as documentation for that process and communicate necessary information to stakeholders such as JackTrip Labs and onboarding developers.

5.4 Backward Compatibility of a Network Protocol

Due to the academic origin and long history of the OSS project there are aspects related to prior decisions that must be considered when assessing the state of the codebase. As indicated in subchapter 4.3.2, before 2004 there was no community-based version control system available for JackTrip development. The knowledge about bugs was limited to a small group of researchers and a single person worked on the project for the most part. Several rewrites introduced the usage of JACK and Qt or urged a redesign to extend features such as the hub-and-spoke operation mode described in the JACKyfication section of subchapter 4.1.2. At that time, those technologies and engineering approaches enabled the project to focus on the problem domain by leveraging existing libraries to create solutions for the challenges in its environment. Later development built upon prior design decisions, while cross-platform audio APIs emerged, and the capabilities of the programming language changed with new C++ standards.

One explanation focusing on specific technologies is to guarantee backwards compatibility as JackTrip implemented a network protocol at a time where streaming applications were rare, and the capabilities of the Internet were limited for realizing networked music performances. For example, since JackTrip development started IPv4 has been prevalent on the Internet as IPv6 has not been widely adopted by Internet service

providers. For that reason, IPv6 could not be considered for practical usage when the protocol was designed. As a result, the elaborations of IE4 in subchapter 4.3.6 describe the challenges to working around the limitations of the protocol in an IPv6 context to implement new features on the one hand and simultaneously provide backwards compatibility to keep the integrity of the protocol between different JackTrip versions on the other hand.

Thus, backwards compatibility seems to be an architecture characteristic that is crucial for the current project. Additionally, all the interviewees agree to some degree on the consequences of complexity and express the desire to start from scratch with a version 2.0. As described in subchapter 4.3.4, this is also related to the idea of moving the core technology into a library that can be used by end applications that implement the operation modes and specify OS details, such as graphical frameworks and audio APIs. IE1 presents his HackTrip implementation described in subchapter 4.3.5, as a new base and IE3 considers it for refactoring. An architect would carefully trade off and take other characteristics, such as scalability, mentioned by IE3 in subchapter 4.3.4, into account. Though the frameworks used in the HackTrip implementation are more modern, as it has been written from scratch, other architectural considerations are not transparent to other contributors. Due to the structures of the academic environment, in which the new project has been developed, there is a probability of similar threats for AE.

5.5 The Legacy of a Research Codebase

Following the arguments in subchapter 2.1.3, the academic origin and impact of research funding may be another reason for focusing on the problem domain while neglecting software development practices for reducing AE. The situation of PhD students temporarily contributing to the JackTrip project to advance a research topic is explicitly mentioned by IE1 and mostly applies to the development period before 2011. However, architectural decisions that have been made by those actors still shape the codebase due to the nature of such a large, long-lived academic research tool suite (3L-ARTS). When considering the project history in subchapter 4.1.2, enriching it with details from the interviews and relating it to the theoretical background, the JackTrip project appears to be an archipelago as described in the archipelago model section of subchapter 2.3.4.

In the early development stage, the objective of the project was to research Internet acoustics in a P2P fashion by leveraging a typical client/server model, as described in subchapter 4.1.2. This represents the original project O in the archipelago model and is referenced as JT-O in the following. Further research and application urged a redesign and introduced JACK and Qt in order to run two JackTrip instances and mix audio streams for conducting performances at three sites. This development path can be interpreted as a first major extension to JT-O and is designated as JT-OM_2 in the archipelago model classification. In the following, the hub-and-spoke modes extensively changed the project and indicated another outcrop named JT-OM_2M_2 as existing APIs needed to be adapted once again to allow multiple client connections and audio mixing.

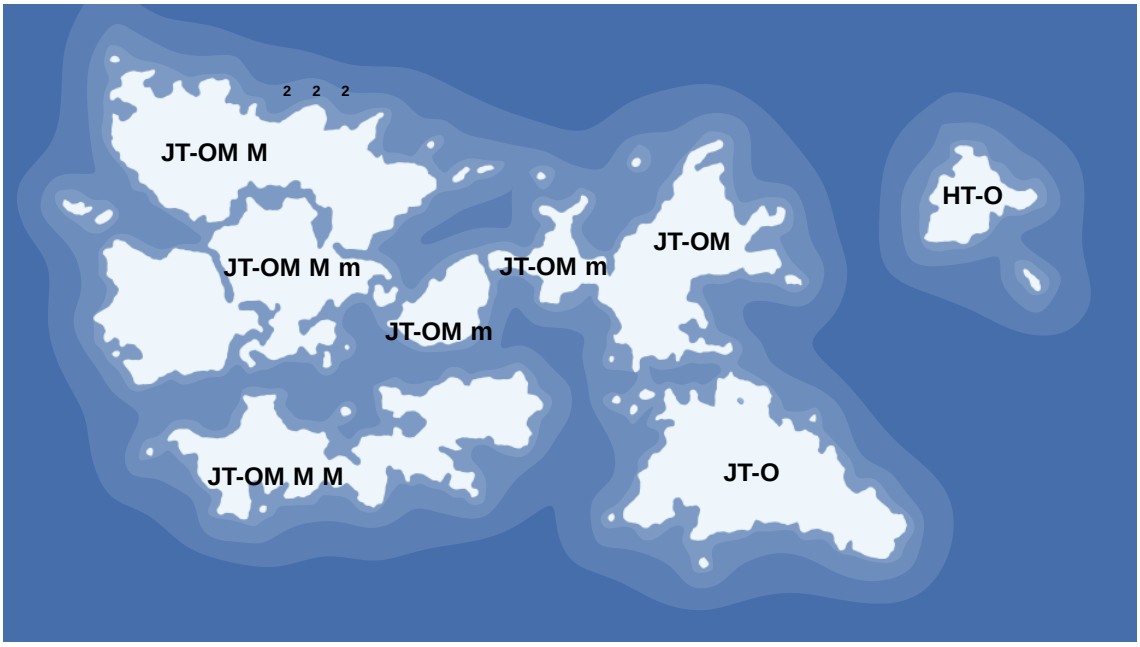

Figure 27: Archipelago model for JackTrip's architecture from version 1.1 to 1.6.8

Minor extensions, such as small features added during further research and application as elaborated by IE1 and IE4 in subchapters 4.3.8 and 4.3.4, respectively, would represent either JT-$OM_2M_2m_1$ or JT-$OM_2M_2m_2$, depending on their usage or adaptation of original APIs. Examples include several adaptations to work around limitations of the protocol as described by IE4 (subchapter 4.3.6). Additionally, the JackTrip Labs Virtual Studio development suggests its own requirements and can be regarded as JT-$OM_2M_2M_1/M_2$ or JT-$OM_2M_2m_1/m_2$, depending on whether the added extensions are major or minor and whether JT-OM_2M_2's APIs enable it directly or have to be adapted. Fig-

ure 27 shows the major and minor extensions as single outcrops in a set of islands according to the archipelago model. In this illustration, HT-O designates the more distant HackTrip rewrite, which so far has not been used as development base.

Consequently, one and the same software project incorporates different concerns to serve different purposes for different users in different contexts. Architecture decisions that have been made by prior contributors tend to follow different objectives and may be incompatible with later development. For example, IE2 highlights that the decision to instantiate JackTrip through the settings class works for a CLI-only application, but complicates the management of settings parameters when introducing GUIs, as discussed in subchapter 5.3. Another question to this aspect is, whether such design decisions happen consciously or unconsciously and whether there is a relation to their accumulation, which is regarded as a symptom of AE. The results do not provide insights into this. However, there are indications of a lack of developers' awareness of AE capable of explaining an unnoticed persistence during the evolution of the SA.

5.6 Contributor's Motivation and Awareness

The developers of the JackTrip project are driven by different motivations. Academic contributors use its codebase for further development to advance a research topic in the problem domain or add features such as a GUI and IPv6 support to ease usage for students and staff. Simultaneously, professional actors create a business model based on the JackTrip application to offer services for NMPs, focusing on customer needs.

In the beginning, all of those stakeholders usually begin contributing in their spare time and for their own needs, which is enabled by the nature of OSS and desired according to the OSD. The transformation into co-developers, as explained for Fetchmail in subchapter 2.1.2, is also typical for a “bazar-style” of development. This implies that most contributors will necessarily begin to engage in the project as a hobby. In subchapter 4.3.4 IE2, states that he became involved that in way and points out that other contributors introduced important maintenance considerations and single features, though some of them left the project thereafter. In this regard, a certain lack of long-term commitment applies to JackTrip's developer community as it does for other OSS projects and may be one of the reasons for AE, according to the theoretical background in subchapter 2.3.4.

Furthermore, in subchapter 4.3.8, IE1 highlights the fact that in order to approach the problem domain, contributors need to evaluate the results as musicians. Thus, the practice of a musician is as important as that of an engineer, and using trained ears for audio quality evaluation is preferred over approaching JackTrip development in an abstract way. This corresponds with the culture of multivocality that is typical for CCRMA and has justified new disciplines such as computer research in music. Absolute partitions into one or another discipline cannot foster qualifications at their intersections. Similarly, as researchers who need to split time between software development and research activities (see subchapter 2.1.3), “computer musicians” (Interview #1) may also face the requirement to allocate time for activities in both challenging domains. The statements of IE2 and IE4 in subchapter 4.3.8 support that argument in that onboarding contributors are typically musicians and eventually become developers, which may indicate a latent inexperience.

However, other aspects in relation to a potential lack of developers' awareness do not appear in the results. In contrast, IE4 explains that architectural considerations are taken into account, as described in subchapter 4.3.5, and apart from the Qt controversy, there is no statement about improper or unsuitable code. As the investigations in this work focus on architectural aspects of higher structural order, code-level metrics for supporting that argument are not available. Likewise, the results do not allow assessing the sufficiency of the business knowledge of the developers. As elaborated by IE3 for JackTrip Labs (subchapter 4.3.3), the project management and organization of the development process indicate that full-time developers follow agile methodologies according to the best practices outlined in subchapter 2.2.7 to react to changed requirements and cope with limited resources and time constraints.

5.7 Professionalization of the Project

Concerning the project management, a reorganization of the contribution practices has been necessary to avoid out-of-sync branches and related implications for merging code. There was a broken release, that was fixed very quickly, as IE4 remarks in subchapter 4.3.2. Moreover, the addition of new features without subtracting earlier ones, such as the introduction of multiple GUIs, complicates navigation in the ever-growing code-

base, according to IE1, and triggers clashes in IE2's opinion (see subchapter 4.3.1). The reasons for such activities may be related to time pressures and recurring changes. The former is constantly perceived by IE3. However, the aforementioned methodologies actively support reprioritizing tasks and scheduling feature development. Contrary to typical reasons for the time pressure symptom, workloads are not changing, and the bug fixes always take precedence over new features, according to IE3. Though this does not completely guarantee the avoidance of temporal solutions that lead to architectural debts in the codebase, developers at JackTrip Labs “try to test things as much as possible” (Interview #3), keep track of issues, and investigate their origin for mitigation.

Recurring changes are another symptom that is highly related to making mistakes in the development process, as described in subchapter 2.3.4. The number of contributions by JackTrip Labs affects the ability of IE2 to overview the codebase and keep track of changes (see subchapter 4.3.1). He also regards the introduction of the Virtual Studio GUI in addition to the existing Qt one as problematic. This may be an example of how adding features without subtracting earlier ones and non-compliance with former design decisions impair maintaining a manageable codebase. As described in the archipelago model (subchapter 2.3.4), the consequences of neglecting or violating such boundaries can include a range of dissimilarities in the architecture. However, since there do not seem to be any records of decisions and their rationale, it is difficult to follow the implicit guidelines that would be necessary to avoid ongoing AE.

Though the results do not provide insights into whether the broken release arose from frequent changes or if other factors caused such a situation, there is a relationship to this symptom. In subchapter 4.3.7, IE4 mentions that initial considerations and conditions of the project urged focusing on latency minimization while neglecting features of a user-friendly interface. As technology has improved drastically, and modern frameworks support latency reduction, JackTrip Labs' objective may be to further improve usability, which will result in a trade-off with low latency ideals. In order to find a compromise, the codebase changes continuously as software engineering is a practice of iteration. Between both extremes an architect would mediate and provide data from analysis to support, finding optimal decisions in the long run. He or she would also take care of documenting and maintaining the SA from a holistic point of view, as explained in this work.

6 Conclusion and Outlook

In this work, the JackTrip project was investigated within a case study on prominent reasons for AE in OSS originating from academia. A mixture of qualitative and quantitative methodologies were employed by using a broad variety of data sources to answer the research questions.

The first research question (RQ_1) aimed at assessing trends in structural symptoms for AE in this contemporary project through a static code analysis. Using ACDC and PKG techniques for architecture recovery and applying architectural metrics and smell detection to those views revealed a constant increase in link overload smells (11 as of version 1.6.2) and a dependency cycle that affects an increasing number of clusters in the evolution. This confirms the first research hypothesis (RH_1), which states that the chosen OSS academic project might unnoticeably accumulate an AS number that keeps increasing during its evolution. However, the results must be considered with care according to ARCADE Core's author. PKG has been primarily designed for Java systems using a modular directory structure that is not mandatory for C++ systems, and for ACDC, the smells and metrics are determined by the structure of the system dependencies, revealing any significant subgraphs and their degree of coupling. Each recovery technique represents only one architectural view at a time as it can not include various conflicting views in a single model. Using two different recovery techniques yielded different results, which had to be interpreted in context. Thus, structural issues found and described in this work should not be considered as categorical statements about JackTrip's SA.

In addition to ACDC and PKG, the chosen ARCADE Core tool employs other recovery techniques, such as architecture recovery using concerns (ARC) with Jensen-Shannon divergence or limbo similarity measures, to create another architectural view and detect, for example, concern-based smells. Using such a recovery technique in further studies may provide a supplementary perspective on the architecture of JackTrip. Additionally, code-level metrics could support in tracing AS types within the code, revealing issues that do not directly induce or contribute to AE symptoms, but only appear at a higher structural level. Further studies may also investigate other issue indicators in relation to the results of the thematic analysis of interviews in this work.

RQ_2 focused on identifying to what extent prominent causes for AE in OSS apply. Considering the architectural metrics results showing an increasing number of smells in combination with the statements of interviewees, it can be concluded that JackTrip has clearly been subjected to a rushed evolution since the pandemic in 2020. JackTrip Labs seem to impact that change of speed most significantly in system evolution. The scope of their contributions and their influence on the OSS project are controversial. Differing perspectives on using the JACK and Qt frameworks and adopting new C++ standards among JackTrip Labs and developers in the OSS seem to collide in the project. Examples elaborated in the Discussion (Chapter 5) raise the question of how architectural decisions are recorded and who is actually architecting the broader project, holistically. Further studies might focus on these questions and include other data sources, such as the mailing list[45], to investigate implicit architectural decisions in conversations.

Regarding time pressure and recurring changes, there are implications arising from JackTrip Labs' participation that are primarily indicated in the interviews. Though they follow best practices for the project management and development process, there have been situations of clashes in the codebase leading to a broken release. Additionally, disregard for prior design decisions has rendered the codebase difficult to manage. Whether these issues are directly related to the reasons mentioned at the beginning of this paragraph or a consequence of lacking architectural documentation and its management and communication, for example via ADRs, would have need to be investigated separately. In addition, time constraints promote practices that deteriorate the codebase over time; these originate from organizational aspects in academic environments and funded research. One example is the temporary contribution of PhD students advancing a research topic and lacking time to learn and apply software development practices for reducing AE. With that said and taking the present rushed evolution into consideration, the findings strongly indicate the opposite of the research hypothesis (RH_2), which stated that in academia, rushed evolution, recurring changes, and time pressure occur less than in professional environments, making other reasons for AE more plausible. The investigations in this work showed that academic OSS projects face time constraints for similar reasons related to the organization of funded research.

45 http://groups.google.com/group/jacktrip-users

Another more plausible reason, as assumed in RH_2, is a lack of developers' awareness about SA, which has also been researched in this work and resulted in the following distinction. On the one hand there are voluntary contributors only committing to a single feature or using the codebase to advance a research topic in an academic setting. On the other hand, professional stakeholders shape the project to run a business based on the capabilities of the JackTrip application. Due to the nature of an OSS project all of these participants start contributing voluntarily and vary in terms of their individual long-term commitment, which may change in the course of their contribution. Thus, an absolute statement about the long-term commitment of all contributors is not feasible, though it may apply to certain contributors in the overall system evolution.

Likewise, evaluating the development experience and business knowledge of contributors in OSS may distort the perspective on the project for the same reason. The results of this work may show a potential tendency towards attracting inexperienced developers in the first place, but they also show that the reasons may be related to the necessity of devoting time to other activities. However, assessing the experience of developers in an OSS project would require a broader investigation and much more data out of the scope of a master's thesis. Furthermore, an evaluation of such qualifications would only a snapshot and would not necessarily reflect the state of the codebase due to the constant evolution in the practice of software development, which also applies to developers' skills. An absolute statement about developers' experience in OSS is destructive in the discussion about reasons for AE undermining the fact of constant change, as described regarding developers' commitment in the previous paragraph.

The accumulation of design decisions as a reason for AE could not be determined explicitly in this work. Though JackTrip has gained many features and evolved from a CLI-only and peer-to-peer application to an extensive solution for networked music performances comprising different operation modes and GUIs, the system is still evolvable and not considered to be difficult to maintain from the perspective of at least half of the developers interviewed. The backward compatibility of a network protocol and the legacy of a research codebase, as discussed in Chapter 5, suggest that there are conditions leading to issues related to the accumulation of design decisions. Further studies may examine this aspect in more detail.

Ultimately, the results answer RQ_3 regarding how the academic environment and problem domain relate to the symptoms and reasons for AE in JackTrip. Detailed information on the origin and early development was analyzed in examining the project's nature between research and application. JackTrip, as a CCRMA project, embodies the culture of multivocality. Its contributors identify themselves on a continuum between software engineers and artists. Thus, role hierarchy and project management may not apply as strictly as in professional software companies. This means that some developers engage more in single technical domains, while others pursue a more generalist approach.

Furthermore, the time constraints of funded research force academic contributors to JackTrip to prioritize advancing in the problem domain over learning and software development as opposed to professional practitioners. This is an organizational issue originating from academic structures and independent of the particular project, institute, and qualification of contributors. With that said, RH_3's statetmen that domain-specific circumstances and the academic environment shape the development approach in the OSS project and impact the contributor selection can be confirmed.

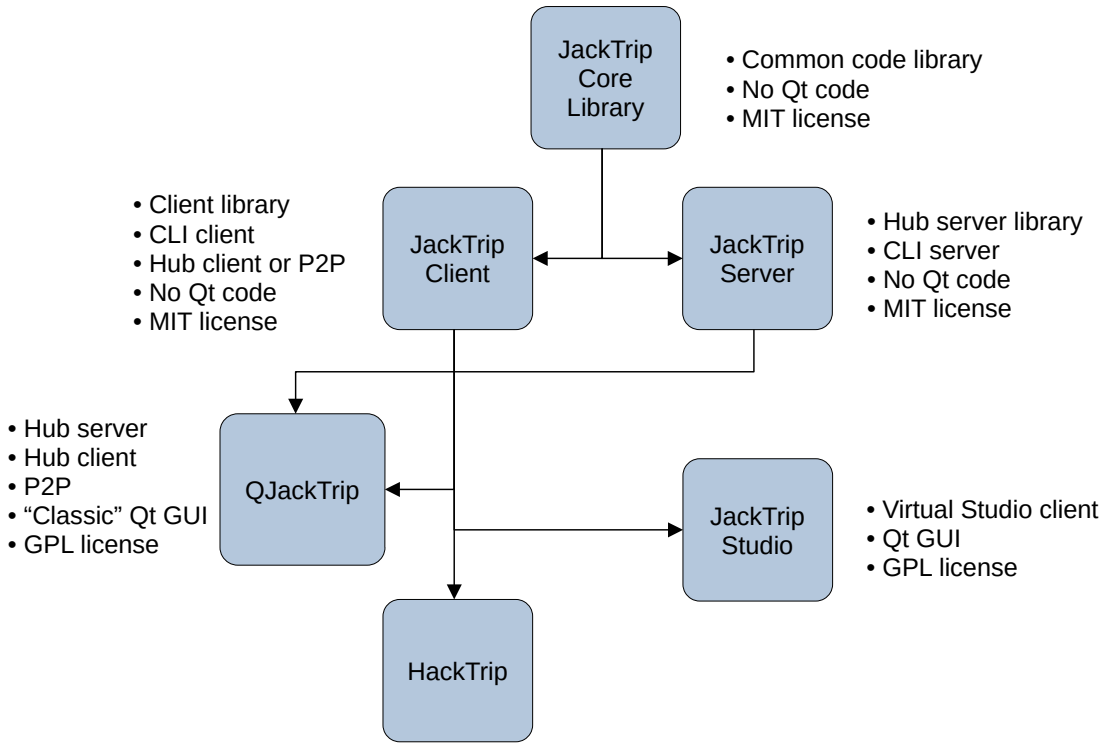

Figure 28: Prospect on JackTrip's potential architectural redesign as of version 2.0

To conclude this work, the main issues arising from the aforementioned aspects are a missing designation of architects and architectural work, as described at the beginning of this Conclusion and Outlook. Moreover, the codebase is comparable to an archipelago of specific applications and use cases that are implemented by a single system. However, all the interviewees agree on changing that design in the future by separating the core technology into a library that is used by single end-applications. This prospect is shown in Figure 28. Similarly, a protocol specification that is versioned is being discussed to mitigate current limitations and related workarounds. Using the presented methodologies and insights, the JackTrip project might introduce a process for using software architecture recovery (SAR) in circulation with software architecture analysis (SAA) and technical debt (TD) removal on a regular basis to integrate countermeasures for improving the architecture during release cycles, as shown in subchapter 2.3.5 Analysis and Recovery of Software Architecture.

7 Bibliography

Azadi, U., Fontana, F. A., & Taibi, D. (2019). *Architectural Smells Detected by Tools: A Catalogue Proposal*. IEEE. https://doi.org/10.1109/TechDebt.2019.00027

Baabad, A., Zulzalil, H. B., Hassan, S., & Baharom, S. B. (2020). Software Architecture Degradation in Open Source Software: A Systematic Literature Review. *IEEE Access*, *8*, 173681–173709. https://doi.org/10.1109/ACCESS.2020.3024671

Bosi, M., Servetti, A., Chafe, C., & Rottondi, C. (2021). Experiencing Remote Classical Music Performance Over Long Distance: A JackTrip Concert Between Two Continents During the Pandemic. *Journal of the Audio Engineering Society*, *69*(12), 934–945. https://doi.org/10.17743/jaes.2021.0056

Cáceres, J.-P., & Chafe, C. (2010a). JackTrip: Under the Hood of an Engine for Network Audio. *Journal of New Music Research*, *39*(3), 183–187. https://doi.org/10.1080/09298215.2010.481361

Cáceres, J.-P., & Chafe, C. (2010b). JackTrip/SoundWIRE Meets Server Farm. *Computer Music Journal*, *34*(3), 29–34. https://doi.org/10.1162/COMJ_a_00001

Carnegie Mellon University Software Engineering Institute. (2010). *What Is Your Definition of Software Architecture*. Carnegie Mellon University Software Engineering Institute. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513807

Caulfield, J. (2022, November 25). *How to Do Thematic Analysis: | Step-by-Step Guide & Examples.* Scribbr. https://www.scribbr.com/methodology/thematic-analysis/

CCRMA - Stanford University. (2010). *SoundWIRE Research Group at CCRMA*. SoundWIRE Research Group at CCRMA, Stanford University. https://ccrma.stanford.edu/groups/soundwire/publications/

Chafe, C. (2018). I am Streaming in a Room. *Frontiers in Digital Humanities*, *5*. https://www.frontiersin.org/article/10.3389/fdigh.2018.00027

Chafe, C., & Leistikow, R. (2001). *Levels of temporal resolution in sonification of network performance*. https://smartech.gatech.edu/handle/1853/50623

Chafe, C., Wilson, S., Leistikow, R., Chisholm, D., & Scavone, G. (2000). A Simplified Approach to High Quality Music and Sound Over IP. *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy, December 7-9,2000*, 5.

Coelho, J., Valente, M. T., Silva, L. L., & Hora, A. (2018). Why We Engage in FLOSS: Answers from Core Developers. *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 114–121. https://doi.org/10.1145/3195836.3195848

Computer Science Department University of Southern California. (2017). *ARCADE Manual – Software Architecture Recovery, Smell Detection and Visualization*. https://tiny.cc/arcademanual

Coulin, T., Detante, M., Mouchère, W., & Petrillo, F. (2019). *Software Architecture Metrics: A literature review* (arXiv:1901.09050). arXiv. https://doi.org/10.48550/arXiv.1901.09050

de Souza, M. R., Haines, R., Vigo, M., & Jay, C. (2019). *What Makes Research Software Sustainable? An Interview Study With Research Software Engineers* (arXiv:1903.06039). arXiv. https://doi.org/10.48550/arXiv.1903.06039

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional. https://www.oreilly.com/library/view/domain-driven-design-tackling/0321125215/

Glinz, M. (2008). A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Software*, *25*(2), 34–41. https://doi.org/10.1109/MS.2008.31

Groen, D., Guo, X., Grogan, J. A., Schiller, U. D., & Osborne, J. M. (2015). *Software development practices in academia: A case study comparison* (arXiv:1506.05272). arXiv. https://doi.org/10.48550/arXiv.1506.05272

ISO. (2011). ISO/IEC/IEEE 42010:2011, Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, 1–46. https://doi.org/10.1109/IEEESTD.2011.6129467

JackTrip Labs, Incorporated. (2022, Juli 18). *What is the JackTrip Virtual Studio?* JackTrip. https://help.jacktrip.org/hc/en-us/articles/1500004365742-What-is-the-JackTrip-Virtual-Studio-

Koschel, A., Rausch, A., & Gharbi, M. (2019). *Software Architecture Fundamentals*. https://www.content-select.com/index.php?id=bib_view&ean=9783960886440

Laser, M. S., Le, D. M., Garcia, J., & Medvidović, N. (2021). *Architectural Archipelagos: Technical Debt in Long-Lived Software Research Platforms* (arXiv:2104.08432). arXiv. https://doi.org/10.48550/arXiv.2104.08432

Le, D., Link, D., Shahbazian, A., Zhao, Y., Mattmann, C., & Medvidović, N. (2017). *Toward a Classification Framework for Software Architectural Smells*. https://www.semanticscholar.org/paper/Toward-a-Classification-Framework-for-Software-Le-Link/4baefec14c6d4d4a170df7243f5a9ff5cf51a960

Le, D. M., Behnamghader, P., Garcia, J., Link, D., Shahbazian, A., & Medvidovic, N. (2015). An Empirical Study of Architectural Change in Open-Source Software Systems. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 235–245. https://doi.org/10.1109/MSR.2015.29

Le, D. M., Carrillo, C., Capilla, R., & Medvidovic, N. (2016). Relating Architectural Decay and Sustainability of Software Systems. *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 178–181. https://doi.org/10.1109/WICSA.2016.15

Li, R., Liang, P., Soliman, M., & Avgeriou, P. (2022). Understanding Software Architecture Erosion: A Systematic Mapping Study. *Journal of Software: Evolution and Process*, *34*(3). https://doi.org/10.1002/smr.2423

Li, R., Liang, P., Soliman, M., & Avgeriou, P. (2021, März 21). *Understanding Architecture Erosion: The Practitioners' Perceptive*. https://doi.org/10.1109/ICPC52881.2021.00037

Lilienthal, C. (2019). *Sustainable Software Architecture*. https://www.content-select.com/index.php?id=bib_view&ean=9783960887805

Link, D., Behnam, P., Moazeni, R., & Boehm, B. (2019). *The Value of Software Architecture Recovery for Maintenance* (arXiv:1901.07700). arXiv. https://doi.org/10.48550/arXiv.1901.07700

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson. https://www.oreilly.com/library/view/clean-architecture-a/9780134494272/

Mitchell, B. S., & Mancoridis, S. (2006). On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering*, *32*(3), 193–208. https://doi.org/10.1109/TSE.2006.31

Nelson, A. J. (2015). *The Sound of Innovation: Stanford and the Computer Music Revolution*. MIT Press. https://ccrma.stanford.edu/~aj/TheSoundOfInnovation.htm

Open Source Initiative. (2007, März 22). *The Open Source Definition (Annotated)*. https://opensource.org/osd-annotated

Raymond, E. S. (2000). *The Cathedral and the Bazaar*. Penn Libraries, University of Pennsylvania. http://www.catb.org/esr/writings/cathedral-bazaar/

Richards, M. (2018a, März 12). *Lesson 8—Components* [Online Training]. Developer to Architect. https://www.developertoarchitect.com/lessons/lesson8.html

Richards, M. (2018b, August 6). *Lesson 29—Component and Service Coupling* [Online Training]. Developer to Architect. https://www.developertoarchitect.com/lessons/lesson29.html

Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture*. O'Reilly. https://www.oreilly.com/library/view/fundamentals-of-software/9781492043447/

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012a). Background and Definition of Concepts. In *Case Study Research in Software Engineering* (S. 11–21). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118181034.ch2

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012b). Data Analysis and Interpretation. In *Case Study Research in Software Engineering* (S. 61–76). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118181034.ch5

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012c). Data Collection. In *Case Study Research in Software Engineering* (S. 47–60). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118181034.ch4

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012d). Design of the Case Study. In *Case Study Research in Software Engineering* (S. 23–45). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118181034.ch3

Schmitt Laser, M. (2022, November 12). *Release v1.2.0 · usc-softarch/arcade_core*. GitHub. https://github.com/usc-softarch/arcade_core/releases/tag/v1.2.0

Schmitt Laser, M., Medvidovic, N., Le, D. M., & Garcia, J. (2020). ARCADE: An extensible workbench for architecture recovery, change, and decay evaluation. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1546–1550. https://doi.org/10.1145/3368089.3417941

Tonnätt, N. (2021a, März 13). *Build Instructions*. Linux - JackTrip. https://jacktrip.github.io/jacktrip/Build/Linux/

Tonnätt, N. (2021b, März 13). *Resources*. Resources - JackTrip. https://jacktrip.github.io/jacktrip/About/Resources/

Tornhill, A. (2018). *Software Design X-Rays*. https://pragprog.com/titles/atevol/software-design-x-rays

Tzerpos, V., & Holt, R. C. (2000). ACCD: An algorithm for comprehension-driven clustering. *Proceedings Seventh Working Conference on Reverse Engineering*, 258–267. https://doi.org/10.1109/WCRE.2000.891477

Whiting, E., & Andrews, S. (2020). Drift and Erosion in Software Architecture: Summary and Prevention Strategies. *Proceedings of the 2020 the 4th International Conference on Information System and Data Mining*, 132–138. https://doi.org/10.1145/3404663.3404665

Yin, R. K. (2018). *Case Study Research and Applications* (6. Aufl.). Sage Publishing. https://uk.sagepub.com/en-gb/eur/case-study-research-and-applications/book250150

Zirkelbach, C., Krause, A., & Hasselbring, W. (2019). *Modularization of Research Software for Collaborative Open Source Development* (arXiv:1907.05663). arXiv. https://doi.org/10.48550/arXiv.1907.05663

8 Index of Tables

9 Table of Figures

10 Table of Codeblocks

11 List of Abbreviations

Algorithm for Comprehension-Driven Clustering.....ACDC
Application Programming Interface.....API
Architectural smell.....AS
Architecture decision record.....ADR
Architectural erosion.....AE
Architecture Recovery, Change, And Decay Evaluator.....ARCADE
Architecture-to-Architecture.....a2a
Center for Computer Research in Music and Acoustics.....CCRMA
Cluster coverage.....cvg
Cluster-to-Cluster.....c2c
Command-line Interface.....CLI
Continuous Integration.....CI
Domain-Driven Design.....DDD
Graphical User Interface.....GUI
International Organization for Standardization.....ISO
Internet Protocol.....IP
Interviewee 1.....IE1
Interviewee 2.....IE2
Interviewee 3.....IE3
Interviewee 4.....IE4
Jack Audio Connection Kit.....JACK
Lack of Cohesion in Methods.....LCOM
Large, Long-Lived Academic Research Tool Suites.....3L-ARTS
minimum transform operation.....mto
Modularization Quality.....MQ
Networked Music Performance.....NMP
Open Source Definition.....OSD
Open Source Software.....OSS
Package Structure.....PKG
Quality of Service.....QoS
Quality Requirement.....QR
Research Hypothesis.....RH
Research Question.....RQ
Rigi Standard Format.....RSF
Round-Trip Time.....RTT
Software Architecture.....SA
Software Architecture Analysis.....SAA
Software Architecture Patterns.....SAP
Software Architecture Recovery.....SAR
Sound Waves from the Internet from Real-Time Echoes.....SoundWIRE
Technical Debt.....TD
Test-Driven Development.....TDD
Transmission Control Protocol.....TCP
Unified Datagram Protocol.....UDP

12 Appendix

12.1 Interview Transcriptions

12.1.1 Interview #1

Date: 2023-01-20

Participants: Interviewer (IR), Interviewee 1 (IE1)

IR: OK, so the research background is that the software architecture of a system is like a bare brickwork in construction that is carrying the source code. But during implementation, there are trade-offs, and they require you to deviate from such a blueprint. These are just natural and can't be avoided. However, if the planned architecture is violated, then the system becomes hard to maintain and evolve, and it probably breaks. You can imagine that like a bungalow with several unplanned extensions on the roof that are leading to cracks in the building. And in software engineering, that indicates the phenomenon of architecture erosion. And I would like to learn about the JackTrip software project. Its origin, evolution, and environment to find possible reasons for that.

IE1: I've been around this project since its inception, approximately year 2000. And I'll talk about the predecessor, which is called StreamBD. Capital B, capital D, which stands for both directions. So it was a streaming engine for uncompressed audio between two hosts separated by networking. In particular, it was ethernet protocol. And just a word about why this was developed. The original intention was to do streaming between two hosts in geographically separate areas and play with a sort of fantasy notion that I had at the time of, can you use time delay on the network to create a plucked string? Now, time delay goes into any musical instrument that has a pitch, almost any. So a stretched string, you can imagine that in your mind, think of a very, very long string, and you've got both ends fixed. And you bang it on one end, and you watch the wave travel down to the other end. Then it bounces back, and it comes back to where you are, and it bounces back where you are back to the other end. It recirculates, and it goes around and around. And the length of the string determines the timing of that round trip, which governs the pitch. So now in your mind, think of a guitar string and how you finger the string so that it gets shorter to play higher pitches. Well, that's just like mak-

ing a faster round trip. And the time delay of the network typically is in kind of a pitch range that we can hear, which is, I don't know, oddly coincidental, I suppose. And that was the fantasy. Could you actually take those time delays and create this network string and then listen to it? So in order to do that, needed a way to stream sound between two sites and have that recirculate from both ends. And also needed to have a little bit of filtering so that the losses in the string would be simulated in this network circuit. And it worked. It worked great. And we sort of as a sort of byproduct, I guess, in trying this idea out, we ended up with a uncompressed audio bi-directional streaming software, very low latency. And so that was StreamBD written in C++ team effort, definitely, and done at CCRMA. And at the time, I was also working in the Banff Center for the Arts in Alberta, Canada. And a lot of the development was done up there as well. It was kind of nice to be able to go back and forth between Banff and Stanford, California, as setting up machines on both sides, using that as a path. And you have to mention that in the year 2000, this was pretty restricted to being able to do it at universities. You needed what at the time was called next generation networking. Today, I'd say our home networking capabilities are even surpassing what we had available two decades ago in the university setting. But Banff Center for the Arts was on the Canary Network, which is a research network in Canada. Stanford, of course, was on Internet2 in the USA. And we could send audio back and forth between both sites across next generation networks because they appear correctly with each other. So anyway, that's the original project.

IR: Can you tell me about some challenges in that course? Maybe we got some technology challenges, but also in organization and process handling?

IE1: There's stuff on both sides of that question. So technical challenges, first, we needed to decide on which of the network protocols we were going to use for the audio packet streaming. Eventually ended up with a completely UDP-based system. We also needed to interface to the audio drivers on the particular host that we'd be using. So that would be in the early days of Linux, that would have been Open Sound System, OSS. And eventually, also the Advanced Linux Sound Architecture, making connections to that through ring buffers and coming up with a ring buffering structure that was tricky. And then, very important, the threading paradigm. So you have a network thread, you have an audio thread. How do you coordinate the concurrent tasks that are going on?

How do you set the task priorities and that sort of thing? So that was all researched kind of hands-on way with StreamBD. We learned a lot. We learned what libraries to use at the time that were good for that. We used it in, as I said, the Pluck String demo. But we also used it in other kinds of demos. The very first public demo was year 2000 in Dallas, Texas with the Supercomputer 2000 conference. And I set up a four-channel StreamBD in Dallas at the conference site, which connected to CCRMA. I played an electric cello with four strings that had four separate outputs. So each of the strings had its own StreamBD channel. Each of those channels connected to Stanford on four loudspeakers. And the loudspeakers were in a stairwell, big resonant space. And then there were four microphones that were positioned also in the stairwell. And those came back to the conference site in Dallas. And so I was playing the electric cello, reverberating myself in real time in a stairwell in California. And the sound came back to me in quadraphonic. So I had four loudspeakers around the demo area. And so what people heard was this kind of very, very luxurious sounding cello sound with natural reverberation. But it was transcontinental. So that was year 2000. And in order to actually make our connection in Dallas to Internet2, we had to go through the Supercomputer 2000 networking infrastructure, which at the time, they would talk about this was a supercomputer. And they were very much a part of the networking research community. This is where a lot of new techniques and products were developed and shown annually. And so my demo was with people from Internet2, who provided a lot of the local infrastructure. And we set up a test network test bed in the site, where we could turn it on and off something called expedited forwarding, which was a new protocol, ethernet or networking protocol, I should say, being researched by Internet2. And so that was part of the demo. You could hear this cello player. And then you could say, OK, what's it sound like if we turn off expedited forwarding as a bit in each of the packets headers? And you'd hear the difference. There would be more packet loss. So we got an award for this demo. I'm very proud of that. It was called the best tuned and most imaginative award or something like that. Anyway, it was a really fine experience and proof of our StreamBD software. So we were all very happy after that. So that, actually, when I mentioned this turn on and off the bits thing, that has to do with policy stuff. So then the second part of your question was not technical, but what about policies and team and things like that? Well,

absolutely, the access to advanced networking test beds was key for us at the time, a bit unusual and rare. In some cases, we were always working, in fact, with the local networking experts, the gurus for a campus, the people who really know how to wire up your place and get the fastest path going across an institution and open the firewalls, particularly port blocked things that are there for privacy reasons. So all of that stuff meant that there was a lot of teamwork with the infrastructure people in any of these sites. So a lot of testing, a lot of careful planning, a lot of measurement, and a lot of collaboration. I'd say, overall, what I found was the advanced networking people, the ones who were taking care of this infrastructure, were super into the idea. They built this stuff and they wanted to see it used like a race car. They wanted to see where it would go. And our application was one of the more demanding ones. And you would hear the result. If it wasn't right, it would be garbage. And if it was right, it would be a kind of audio experience in telecommunication that nobody had ever had before. So very, very rewarding to put effort into. In some cases, just like in Dallas, the actual physical hardware and the path through it and software had to be custom provided. And this kept being the case in other places. Soon after that, we were doing concerts between universities and finding that each site had its own policies and its own team that you had to bring on board in order to make it work. And then the StreamBD software team, like I said, mostly at Stanford and in Canada, there was an equivalent team working at McGill University in Montreal. And we collaborated a lot, shared notes. Their software was concentrating on video and had also uncompressed audio and different buffering schemes. So we were able to kind of compare approaches, which was really good. And then we did things together because that's a nice transcontinental distance that you could really test things with. And there are research papers written by all of the groups at that time, so there's a lot of documentation.

IR: One question about the other thing you mentioned about the cello concert. Are those artistic and research functionalities still considered during development?

IE1: I don't think you can separate them. That's an important point that you bring up, that you can do a lot of stuff just purely engineering, but you don't know exactly how to evaluate or what the questions are until you put it to use. And the most discerning use is to put musicians in the experiment and have them mess with it and tell you what works

and what doesn't work and suggest. And often I've made the mistake myself of saying, well, we're going to have a population of musicians, we're going to have a population of engineers, and they'll talk to each other. But that's a mistake because a lot of the engineers are musicians, and a lot of the musicians become engineers very quickly. And it's particularly important to have good ears and be able to talk about what you're hearing. And so everybody does naturally have that capability, and they develop it. Whether they're musicians learning to network dropouts or engineers listening to audio quality, all of those skills develop very, very quickly, I think. And so there's a lot of people who worked on the project then and work on it now who are very versed in both aspects.

IR: So can this be related to the onboarding process? Like you told me that the engineers kind of turn into musicians and vice versa. So is this part of the onboarding, like the characteristics to choose the people who are working on the project? Or does this happen rather coincidentally that those people who are engineers happen to become musicians or the other way around?

IE1: I think more it's the coincidentally answer. Absolutely. And I've seen, not in our cases, but similar projects where a search for help goes out to hire someone to work on network audio or network music performance. And it may not work out well if the person doesn't really have the appreciation of audio and audio quality and is doing it kind of in a more abstract way and not using their ears. So it makes a difference. And I think we've been lucky because most people involved were coincidentally both maybe leaning one way or the other because of their past formation, but they may adapt very, very quickly. Often I get the question, what are you? And in my case, I say it's an unfair question because I am a computer musician, I guess.

IR: What about the speed of evolution compared to previous periods? I've seen lately that the releases happen very quickly. I think as of 2019, 2020. How do you experience that? And how does it affect your practice as a developer?

IE1: There's a bunch of periods we could actually chart out to see what the rate of development was and where the major milestones happened. And at this point, like you said, it's going very fast. There's a lot of development. There's a lot of making ease of use features coming together. There's video now. And the pace kicked up really largely

in March of 2020 with the pandemic. Naturally, a lot of interest, a lot of people with time to help. And I think prior to that, it had been kind of a quiet project. A bit of work in 2019, 2020 before the pandemic that happened because I was in Europe and I was working with two different teams. And before the 2019 period, I would say the quieter period happened between, let's see, I'm just going to guess. If I said 2019, probably 2011, 2012 to 2019, so about seven or eight years where this is a very mature code base. People were using it. It did not gain a lot of features in what you would call the kind of core part of the software. I was developing some features that I wouldn't call part of the core at that time, but then did become later. So the mature software was just kind of there and ready to use. And it was being used by a good number of people. But again, it was sort of within a realm of academic application, concerts, and so on. And the period up before that, it would kind of give it a big, big development set of cycles that happened across 2004 to 2009, 10, somewhere in there. And around 2010, a lot of papers written about the core technology. And like I said, after that period, I think I was able to do a lot of things that after that period, I think after the papers were written, things felt pretty mature and stable, and there weren't releases happening. So that kind of charts it out. When I mentioned the audio drivers and the libraries and stuff, there was a big change that I made in 2004. And you can explain, this was for a demo that we wanted to do, which would involve three sites. Now, you remember StreamBD, both directions, was always peer-to-peer to sites. And so for this demo, we were going to do Canada, California, and Montana. And we're going to have three musicians play a trio together. And it was very successful, but in order to make it work, had to come up with a solution to how do you connect three sites when your software is only capable of two sites. And so different ways to do that at the time. It's still peer-to-peer technology, and you could. One way to do it would be to have your network connect to two different peers from your local host and then share everything. All the packets are being combined correctly and stuff at the network layer. But instead of that, I said, oh, there's this new library and protocol for audio called JACK. And that's the JACK Audio Connection Kit. Sort of a geekish thing to have recursive acronyms, but that's JACK. And JACK lets you set up an audio server on your local application that talks to your sound card or your audio interface. And so I said, oh, well, maybe we could just connect the other two sites as peer-

to-peer connections and combine their audio in JACK. And so that's the way that we went. So JACK allowed a machine to connect to multiple sites and then combine their audio. So it's basically having, for a three-way thing, each site is running two JackTrip jobs, if that makes sense, to the outbound. And so we had a triangle, and each site ran two connections to the other two sites. And about the same time, I also brought in the Qt C++ framework and started to use a lot of the functionality in Qt. So those two enhancements, Qt and JACK, then kind of powered JackTrip through the next decade, I would say, at least. And it's still there. I mean, there's still both of those things. There's a lot of other developments that happened. But the name JackTrip comes from that three-way moment, the first three-way, when I ended up using JACK. So it's Jack triple. I had to come up with a name of some sort for the demo, and the name stuck. So it's kind of cool. The better way to do multiple sites, and what we do nowadays more than not, is not a mesh of peer-to-peer two-way connections, like I just described, but instead having a hub with spokes. And so you set up a hub server somewhere, and each of the sites is a client to the hub server, and can have a peer-to-peer connection to the hub. And the hub has a way of spawning the receptacles for those spokes. So that's an addition made to JackTrip by [redacted]. And I forget the exact year. It was before 2010, for sure. And today, JackTrip Labs is all about putting those hub JackTrip processes in the cloud, and geolocating those. And so a big change from that, too. There's another period of development where we started a company called MusicianLink in the later part of the first decade. And MusicianLink was to build hardware devices that would do peer-to-peer connections. So these were called JamLinks. And the JamLink device could do four peer-to-peer connections. And it was early. It was home broadband dependent, largely, whether it would work or not. It worked fine under university situations. It was not using JackTrip. There was a brilliant engineer who did the hardware and did the software, and designed it from the bottom up, and basically did JackTrip functionality in a very, very tight coding environment of his own, with a proprietary operating system and all this stuff. And eventually, there was an attempt to do a software version of that by adapting the JamLink protocol in JackTrip, so that you could have a JamLink device, and it would connect to a JackTrip software somewhere else. And so you'll see some code in JackTrip still that says JamLink mode, but it's unused today. The company

folded, I forget what year, but it was an early version of, I guess, what we saw in the beginning of the pandemic, where a lot of Raspberry Pi devices started to run JackTrip. And those are very JamLink-like, only the software they're running is completely open source.

IR: That's very interesting. Do you think that things like the hub mode are dependent to the use case? So you have to come up with a concept for connecting those spokes and to stay real time in regard to the case. So it would be the other way. If we're not talking about audio, but video, for example, wouldn't it, in that case, not so important to come up with such a concept?

IE1: Gosh, if I understand right, what I would say is the typical thing, and this even is how MusicianLink worked, the typical thing, let's say, is to have aggregation of peer-to-peer connections and have a back-end server that tells each of the clients where their peer is on the network. And then the clients make peer-to-peer connections with each other. And no audio, in this case, is actually handled by the back-end aggregator, the hub. But JackTrip hub mode is different because the audio flows to the hub and gets processed and mixed and then sent back out. And in video, that's less common. I'm not the expert on video, and I'm sure I've heard about certain places where the video mixes like that too. But there's also versions where the back-end just sort of says, OK, A connected to B, B connected to C, C connected to A. And it's a true peer-to-peer mesh. As you develop a hub where signals all are aggregated and mixed like that, then your bandwidth requirements go up, so do your CPU constraints. And you'll see today that the configuration of geolocated Virtual Studio JackTrip hub rooms, that was a long sentence, you'll see that you have parameters that say, OK, how many participants? And if you have the demands of a larger group, that means you need more bandwidth, you need more processing, and you have to scale up the resources on that cloud instance. And obviously, that costs more. The more you demand, the more you have to pay for. Yeah, so you're right. I mean, there's an effect there. And when those resources are not adequate, when they're under-resourced, then you fall out of real time. It's exactly what you were asking.

IR: Can you relate to making mistakes due to frequent changes whenever the system extends and explain a strategy to retain an overall picture? If we are talking about the code base.

IE1: This is more applicable, I think, to the first decade, where the code base, we didn't have a version control system like we do today, not a community-based one. It might have been in subversion. It was before Git. And there was no community code reviewing and contributing going on. It was just research, one person or two people or three at the most, I think. And so you could kind of contain if there were mistakes, everybody would know about them. And you'd be able to identify the problems and fix them within a very small group of people, maybe one person. There were a couple of moments, like I mentioned, 2004 with JACK and Qt. And at that point, I basically made a new version. And we kept going from there. There was a point after that period, another, let's see, was a student, took all of JackTrip and redid a bunch of it. [redacted], another moment where [redacted] did a very large rewrite. And that's how we got hub mode, for example, as I explained. And that was in the first decade. So there were kind of like major feature additions and code revisions going on. One of the things that accumulated is I wouldn't go as far as to say kludge upon kludge, which is a term I use. But kludge upon kludge is like where you have something that was kind of adapted to make it work. And then you don't go back and straighten things out and make it beautiful. You go to the next feature. And then you adapt it some more. And you make it work, right? So I wouldn't characterize JackTrip as having so much of that. But I do see that over time, as these rewrites happen, they were accommodating decisions that were made in the earlier version and having to adapt to them. Not that they were kludges, but that other decisions could have been made if things had been started from scratch, from fresh. But there's been a long tradition of adding things and not exactly subtracting them, earlier things, some of that anyway. And that even comes down to how you name things and how functions call other functions where maybe if you'd done it in the beginning that way, you would only have one function, that kind of thing. So things do accrete. It makes it a little hard to navigate the code today, actually. I'd say it's a very difficult learning process for somebody who's coming into the project fresh. It's hard to know what is the place where you need to go in order to make a change, because there's

so many things going on concurrently in JackTrip. It's a little hard to sort out. And some of the naming conventions don't help. And I don't know if I talked to you about this. Tell me if I did. I probably did when we first met. But I did a rewrite from scratch myself in the summer. And ended up calling it HackTrip. And HackTrip is a JackTrip hub client, but with no code really from JackTrip at all. It's like me trying to do it from scratch, just so I can figure things out and see how this should properly work. And I made a very efficient, more modern version of a JackTrip hub client in that project. So for me, that was a kind of like, get my head around the code base one more time. Or not the code base as much as get my head around the basic operation of this kind of software. And do it from scratch just to understand. And I did not use JACK. I used a different audio backend called RT Audio. I used Qt, but I started with Qt 6 rather than Qt 5. I started with a GUI rather than a command line. So it's an app of its own. And I fully understand it. You know, it's a one-person job. It's kind of like for my own purposes. And it's on GitHub. Now, when's the last time I did that? Well, 2000, what was it, 2000, around 2017. And I wrote an article about this as well. So it is in Frontier's publication. And it's about wide area internet reverberation. So as a pointer to that, it's all JackTrip. And I was using JackTrip to be able to do something like, remember how I talked about the physical model, the plucked string in the early days, right? The idea here was if you have a bunch of sites connected to a hub, pretend each of those connections is a reverberator itself, like a room. It'd have to be multichannel because it has to have different reflections in the reverberation. And you combine several sites. It's sort of like having a palace with different rooms. And you can open and close the doors. And you can hear the acoustical effect of playing in that room. So that took a bit of engineering. And the problem of getting my head around how all the functions work inside hub mode and all the multichannel stuff I was doing meant that I needed a moment. I had to go back and figure out how JackTrip worked again. And this is around then. And so I actually, as part of the article I wrote, I put in breakpoints or, well, print statements. I mean, they could be breakpoints. But I put in print statements at all of the major stages when a connection gets built. And then I explained the difference between peer-to-peer mode, hub mode. And I left all of those kind of points in the JackTrip code. And it's in the code base now. If you run JackTrip in the command line with a minus capital V, big V as in Victor, then it prints the in-

formation that is documented in that article. So you see step one, make a connection with the TCP listener. Step two, claim your audio. Yeah, things like that. So that was another one of these moments where you have a fairly thick software and you're trying to figure out how it works. And maybe every once in a while, you have to go back and look at the theory of operation in detail. And sure, it's all there and you understand how it works because you're trying to do something new and you can't do it unless you really have your head around the whole thing. So whenever I wrote that article and then again when I did HackTrip.

IR: Hmm, one question I'm having is connected to the topic of the evolution speed. What about time constraints during development? Is there any, I've seen that the, I think the version 1.69 is released today, am I right?

IE1: Oh, yeah, we're going to have a 1.7 with video, but I think maybe that's the 1.69.

IR: Yeah, OK. So how about time constraints? Do you perceive any time pressure during development or towards the next release?

IE1: Sometimes the releases are coming because there is some major event happening that needs to be supported. Definitely early in the pandemic, there was a kind of progression of like, we're going to do this because we have this choir or we have this concert or something like that. And even in the earliest days, 2004, like I said, I came up with that very new formulation because we had a demo coming. Even in year 2000 for the Texas thing, we did a push to get to that stage for the SC 2000 convention. In 2001, we did SC 2001, ran JackTrip with 500 channels to supercomputer thing in Denver, Colorado. And so we certainly pushed hard to get to that. Things can be driven by, let's say, external conferences, demos, concerts. And today, the push, often it's kind of a tandem thing between the open source JackTrip developments that are affected by the commercial Virtual Studio stuff. And there'll be kind of like a back and forth. A new idea for the open source might immediately cause something to release in the virtual studio world and vice versa. The video stuff implicates some changes to the core. And I think 1.69 is going to incorporate exactly what I'm talking about there. So there's a kind of back and forth going on that's very constructive, actually. Is there external stuff like what else would drive timing like that? It's a good question. People finishing research projects,

finishing PhDs, definitely have had an effect. And on the commercial side, probably demos for that side too, I would think, and shows. There have been a couple of big shows for virtual studio. It's called NAMM, National Association of Music Manufacturers. And there was a really fine presentation of JackTrip, I think it was in the summer, this last summer, early in the summer in Anaheim, California.

IR: So is there a change in the workloads connected naturally?

IE1: Yeah, the workload. That's a kind of project management question. So it would be different in the commercial context versus the volunteer code contributors largely on the open source side. So for a lot of people, it's a passion sort of driven by their passion. And they may have competing things in their work, so you never know. And that's very common in open source, I think. And then when the ball gets rolling, a lot of people get inspired. And the ball has been rolling nicely the last couple of years, I would say. And so many features have been added. There are sometimes just really good ideas that then promote a lot of code development, because, hey, let's do this thing. It sounds really good. And I'd say JackTrip Radio is one of those. And definitely the addition of the GUIs that happened. And now the video. So yeah, major inflection points with implications for project management, no doubt.

IR: The second last question is a bit technical. And I understand if I don't ask the question the right way. But when you're discussing the parts of the system during development, are you considering how to create boundaries and connections? So you can have cohesion and coupling happening in the source code?

IE1: Yes, and it is actually very technical, because there's a choice of signaling mechanisms that go between modules. And this was one of the reasons for adopting Qt early on, is that Qt gives you event driven signals and slots. And it's a fantastic mechanism for concurrency. And when I redid HackTrip, I was able to take advantage of the latest in that stuff. So I find that it's fantastic. You can do that now without Qt, but that rewrite hasn't happened.

IR: Is somebody governing things like that?

IE1: Only, I mean, making a rewrite to use the more standard signals and slots these days, it's just a thought. Nobody's doing more with that. But we do use signals and slots like crazy.

IR: OK, one last question. And I see we did some more time. I hope that's OK.

IE1: Yeah, it's OK. Just pretty quick though, yeah.

IR: OK. Imagine what would be different if the project started today?

IE1: Yeah, well, this is part of why I did HackTrip for myself. So I want to answer that question. And one of the key features that you would like to have is that the core technology would become a library with an API rather than a monolithic standalone application like it is right now. So HackTrip is able to be compiled as a library. And then you can incorporate it in other projects. So it's easy to make it part of another standalone application. It's also easy to convert it into a loadable module so that you can use it as a plug-in. And I went from HackTrip to a library. I also went from HackTrip to a plug-in for the Chuck language, which ChucK plug-ins are called Chugins. And so there's a ChucKTrip, which is a Chugin to ChucK, if that makes sense.

IR: That makes sense. OK, good. Yeah, I think that's it. Thank you very much.

12.1.2 Interview #2

Date: 2023-01-24

Participants: Interviewer (IR), Interviewee 2 (IE2)

IR: The Software Architecture of a system is like a bare brickwork in construction that carries the source code, but during implementation trade-offs require deviations from such a blueprint. These are just natural and can't be avoided. However, if the planned architecture is violated, the system becomes hard to maintain and evolve until it probably breaks. Imagine a bungalow with several unplanned extensions on the roof leading to cracks in the building. In software engineering that indicates Architecture Erosion. I'd like to learn about the JackTrip software project, its origin, evolution, and environment to find possible reasons of that phenomenon. How long and for what motivation are you contributing to JackTrip?

IE2: I started contributing to JackTrip when [redacted] was guest professor. The seminar was about networked music performance. So we developed the SPRAWL system that was based on JackTrip. And then the pandemic started and I had a lot of time to just work on a project and just slipped into JackTrip development.

IR: Please tell me about challenges in the project organization, processes and tooling, but also with technology adoption and infrastructure, for example.

IE2: So the biggest problem when I started was that JackTrip is a really old project and was started before C++11. And a lot of modern principles when developing C++ weren't used. And so in the beginning, I did a lot of maintenance work. And to get a sense of the structure of JackTrip, I implemented the Meson build system for JackTrip. Although JackTrip is written in Qt, or with Qt. It was even written in Qt before it got a graphical user interface. So JackTrip at the beginning was just a command line interface. I think a lot of the problems with managing the codebase in JackTrip started afterwards with the introduction of graphical user interfaces.

IR: So you refer to a new feature, or let's speak of a component that has been introduced and that made the codebase a bit diverse, more diverse?

IE2: For example, the structure for creating the JackTrip connection was that you have a settings class. And you put your command line flags as parameters, arguments for when creating settings instance. And then the settings class creates the JackTrip instance, depending on what arguments you used. And maybe, I think for a command line interface, it doesn't make a big difference if your settings class creates the JackTrip instance, or you have a settings or options class that you put into a constructor of your JackTrip class. So because a command line interface just, you do it just once. So where this part of the code lives is not a big of a deal. It might confuse in the beginning if you never saw the codebase. But it's not a problem when you work on the codebase or develop new features. But when you have a user interface where you can change settings or you stop and start JackTrip connection, server or client, then this approach doesn't work. So managing the settings for the GUI, there are actually two different graphical user interfaces, one for the Virtual Studio, and another one, what was QJackTrip. And

QJackTrip was integrated into JackTrip. So because of this approach, you have the management of your settings at two different places in the JackTrip codebase.

IR: Okay, and what about organizational aspects? Like, how does the on- and offboarding of developers happen?

IE2: I mean, like the core team of JackTrip developers. When I started, there were only open source developers, not open source developers, but people that just did it in their spare time or in an academic environment. And so people develop JackTrip for their own use, for concerts that they did or there weren't any customers. So for me, JackTrip development became something like a hobby that I did. So that was the reason I did also maintenance of the project, cleaning things up. And then [redacted] came, who did a lot of CI work and creating GitHub actions to check if JackTrip builds on Windows, Linux and Mac OS. He stayed. And so there were other developers that just developed one feature and then left again. So that's kind of easy to do for the open source project.

IR: What could be barriers for engagement?

IE2: The biggest barrier, is to get a grasp of the code base, how it's organized and if you want to write a feature, what part of the code base your feature should go. There are some things that are just intertwined, like, I mean, the settings thing that I mentioned, but also the audio backends that are used. And those depend on the operating system you use. So you find a lot of macros in the source code where you just check for which operating system you compile JackTrip and this looks messy. We started to change this with newer, more recent C++ features by constexpr. Yeah, but that's an ongoing thing.

IR: So next question would be, how are artistic and research functionalities considered during development?

IE2: I mean, JackTrip comes from an artistic background. It's not a software that was developed by network people, but from actually musicians. I mean, there are other projects that have different approaches to network music. But the goal was for musicians to really play like they are in one room. That's the reason JackTrip is developed. You can't just write JackTrip without thinking about the implications for the user. But I mean, it's very important to think about for what kind of musicians you write your soft-

ware or your protocol. For example, JackTrip is capable of transporting ambisonics signals. If you have a third order ambisonics, like a spatialization system, you can use this 16 channel signal with JackTrip without any problem. But that's not the common musician that wants to play with their band over the Internet. So usually, that is one feature that I wrote that you are actually capable of just sending one input channel of yours and getting another number of channels back. I mean, maybe that's a special thing in Germany that your upload bandwidth is so bad that it's a big deal if you send one uncompressed channel or two. And before that, you just had one number for the number of input and output channels. So if you send two channels, you get two back. And if you want a 16 channel ambisonics signal, you also send 16 channels.

IR: Do you experience a change in the speed of evolution compared to previous periods? And how does it affect your practice as a developer?

IE2: The biggest change with the speed of development, I mean, no, there were two. I mean, the first big change was the pandemic, because the interest in this kind of software just increased enormously. And so that was a big change. And there were a lot of changes in regard to usability. So that was the reason QJackTrip, the first graphical user interface was integrated by [redacted]. The second big change was when JackTrip Labs became bigger and contributed more to the JackTrip application. They have a website for Virtual Studios that you can create. And so you don't have to host your own server. And when they, in the beginning, they concentrated on those web features and making the onboarding, starting with the JackTrip application easier for musicians that don't use a computer every day or are not interested in technical things like audio interfaces and networking. But it is a big shift from developers that develop for themselves or for the parts of JackTrip they are interested in, to a mode of development where JackTrip is developed for customers that use JackTrip Labs Virtual Studio. And I mean, this is all in one application with the classic QJackTrip graphical user interface and the command line interface. So there are parts that are starting to clash.

IR: So personally, how is your practice affected?

IE2: Since JackTrip Labs contributed so much to JackTrip, to the application, there's so much going on that I am not able anymore to check every change that is done in Jack-

Trip. So before I could, I had enough time and knowledge of the code base that I could just understand every change and that's different now for me. So I don't know the other side of the Virtual Studios. So they connect differently than the JackTrip command line interface or the classical user interface, graphical user interface. Yeah, there are things, parts of JackTrip that I don't understand.

IR: Can you relate making mistakes due to frequent changes whenever the system extends and explain your strategy to retain the overall picture?

IE2: That's a difficult question. Can you repeat the question?

IR: Okay, first part is can you relate to making mistakes due to frequent changes or recurring changes whenever the system extends? And the second part is explain your strategy to retain the overall picture.

IE2: You always have problems, you always have to learn the changes in the code base of the projects you are involved as a developer. Of course, I can relate to that because it's something that happens every day. But there are structures, routines that help you that you don't do mistakes that end up in a release of JackTrip. So if you do a pull request for JackTrip, you first have those CI checks, GitHub actions that check if JackTrip still compiles on Ubuntu, Windows and Mac OS. That we do a Clang format check, so if the formatting of your source code isn't done well. But the other part is of course that there must be somebody else who reviews your change. So usually if that's a person that worked on that part of the source code, so they just tell you if you didn't understand a change correctly or if there's a better approach to the problem you want to solve. But I mean the CI, the GitHub actions, the checks, they could be improved a lot. So for example, I have a check in mind to check if JackTrip still complies with the protocol that we used before, so if you do changes and JackTrip doesn't behave anymore like old JackTrip versions, they wouldn't connect anymore. And if this is released, that's really bad. Right now the JackTrip protocol is so simple that it's not very likely that this happens. But if you work on the protocol itself, for example for the change for having a different number of input and output channels, I had to change the meaning of JackTrip header, protocol header. So that was difficult because we couldn't break the protocol because then people that use old versions wouldn't have been able to connect anymore.

IR: To which extent are time constraints influencing your work as a developer?

Well, the last half year, I wasn't very involved in JackTrip development anymore. I mean, that will change, but I will see what part of JackTrip I will develop in the future. I don't know.

IR: So is the workload changing? And how do you cope with it?

IE2: I mean, there are different kinds of tasks in such a project. So you have maintenance tasks that sometimes can be automated, so you don't have to manually do your release builds, for example. If there are changes for JackTrip Labs' Virtual Studio, I don't have to bother with this. So that's not the part where I'm involved. But sometimes there are parts of the codebase where people think it might be a good idea if I look into it and they just ask me if I could do a review or give my opinion.

IR: Have there been design decisions that rendered parts of the system hard to maintain or evolve from your perspective?

IE2: The protocol itself. What I said, the JackTrip protocol is extremely simple, and if you want to extend on this, there's no path for this. So there's no version indicator in the protocol header that you could check or something like what features the other side is capable of. So that's very restricting. So for example, if you would want to add OSC channels or MIDI channels to JackTrip, that's nearly impossible with the JackTrip protocol right now. With the protocol itself. So you could develop a parallel system that works independent of the audio JackTrip protocol. But I don't think that's a very good design if we would do this this way. That's a part I really want to change, to check where the boundaries of the JackTrip protocol are right now and how we could create a new JackTrip protocol that is versioned, that would be capable of having different channel types like OSC and MIDI and maybe even video and prioritize between those different channels. Because the most important part is still the audio. And if you send additional data that has the same priority, then you could get problems with the audio signal. So that's the reason I want to have this all in one protocol.

IR: When discussing parts of the system, are cohesion and coupling considered for creating boundaries and connections, according to design principles?

IE2: I don't think so. I think it's, we don't have a code style. I mean, not the formatting, but the style, how you should write classes or something. And I think it's because JackTrip is so old, you see a lot of different styles and more recent code has a better style, just because it's new and C++ developers learned a lot in the past decades. So, yeah, but maybe you have some other ideas about what cohesion and what was the other term?

IR: Cohesion and coupling.

IE2: I mean, the settings problem, the settings class creates the JackTrip instance. That's bad coupling. That's not good. But yeah, it's worked to split those two things again. And yeah, maybe it's easier to write a new command line interface JackTrip client.

IR: Last question. Well, imagine what would be different if the project started today.

IE2: We would start with a JackTrip library that really just is concerned about the protocol, the network protocol and doesn't mangle with the audio backend or the buffering or the audio side of the buffering. And then, if you have a JackTrip protocol library, you could write single clients for command line interface, for text user interface, for graphical user interfaces or for special clients for the Virtual Studio, for example. Or, for example, you want to use a totally different framework on Mac OS, on Windows and Linux. So, I think, in my opinion, those cross platform approaches are usually bad. They make maintenance more difficult. And, of course, the common parts of your software, if they actually should do the same, should be in a common library that is used by every client. But I don't like the approach of cross platform graphical user interfaces, for example.

IR: Okay, I think that's it. Thanks a lot.

12.1.3 Interview #3

Date: 2023-01-30

Participants: Interviewer (IR), Interviewee 3 (IE3)

IR: First let me explain the background and then we start right away with the first question. The software architecture of a system is like a bare brickwork in construction that carries the source code, but during implementation there are trade-offs that require deviations from such a blueprint. These are just natural and can't be avoided, however if the

planned architecture is violated, the system becomes hard to maintain and evolve until it probably breaks. Imagine a bungalow with several unplanned extensions on the roof leading to cracks in the building. In software engineering that indicates architecture erosion and I'd like to learn about the JackTrip software project, its origin, evolution and environment to find possible reasons of that phenomenon. And the first question would be how long and for what motivation are you contributing to JackTrip?

IE3: I first learned about it a little under two years ago and the way I found out about it was early on in the pandemic when everybody was unable to meet in person in order to play music together. My son is a member of a boys choir named Ragazzi Boys Chorus and he'd been a member of that for many of his years, about half of his years at that point. I was on the board of the organization and I had seen how hard it was for everybody. I saw the effect it was having on my son and I was afraid that it would impact his love of music and his willingness to continue forward as part of the organization. Especially it became clear that it wasn't a short-term thing, but it was going to be a long-term impact. I felt like I had to do something. So my background is in technology and I'm an entrepreneur, I've built and sold three companies before JackTrip Labs. I was working at an enterprise software company at the time and I just started spending my free time looking at what was available, what was possible and trying to learn as much as I could because, like most people at the time I thought, well, why can't you just do this on Zoom or why can't you just do this on any other collaboration technology. I learned quickly about all these limitations and I also had a very strong background in performance engineering. It just intuitively clicked what the problems were and how to go about addressing those and trying to solve them. So the latency issue just became very clear and I looked at how to optimize that path, like what are the various pieces along that path and basically broke it down. I think there's still a page on the website that kind of talks about breaking down the latency problem and looking at the time that it takes to do the encoding of the sound waves into digital transmitting it over the network. Over your local network first, then over your internet connection, then over the backbone and then conversely going backwards to the other persons' internet connection, their local network, their audio interface drivers and everything. So I just looked at all those pieces and tried to understand is it possible to get this low enough and I found some research

papers that people had written about it. The first question is what is the threshold, like how low do you have to go in order to be successful at it, if it's not the 200 milliseconds that we get from Zoom. I found some papers that [redacted] [redacted] wrote and some research that he did on that topic that I found really fascinating to read. And I had a background in physics as well, so I liked learning about acoustics and music. And I've always been a huge music lover. So I kind of discovered that whole field of learning in a way for the first time and got immersed in it and tried to figure out how to apply my technology background to solve that. One of them was testing different pieces of software like looking at the software that was out there and I'd found the JackTrip project and Jamulus and JamKazam and a whole bunch of other tools that were out there. Just started testing everything and tried testing both the software and the devices, so really approaching the problem from a test-oriented, test-first kind of mentality and tried to find which ones seemed like they could work in theory or be possible. And I found that the JackTrip software had great latency characteristics. I could test it on my local network and I could see that this is basically like a live monitor. It's less than a millisecond because my network is less than a millisecond. So the software is doing what I want it to do. It's not getting hung up on all the drivers and buffers and everything else that typically happens there. And then the next key thing for me would be, especially because I was working with a boys chorus who had in my son's group about 40 members and a larger choir had over 200 boys. They're used to singing in these big groups on a stage in front of an audience. So what I really wanted was not just to get two people or three people being able to play together, but a large group of people being able to sing together or a particular large group of boys, initially. So the next thing I wanted to understand was scalability and see how far I could push those packages. I found JackTrip is the only thing that could scale. Everything else either wasn't designed to scale from a multi-threading architecture perspective or it just could never scale like peer-to-peer based approaches, because they hit other walls very quickly. That's when I kind of doubled down. It's like okay I'm gonna focus on JackTrip and really digging in understanding optimizing, and using this to plug it into the boys choir and get them up and running.

IR: Okay, very interesting. How about challenges in the project organization, processes and tooling but also with technology adoption and infrastructure, for example?

IE3: There are a bunch of challenges. The initial challenge was that JackTrip was command line based at the time. So I had to do everything through a command line and only a very, very small population of people are comfortable using the command line. I think everybody's able to use it, but I think that more often than not people who are able will shy away and just refuse something even if it looks a little too technical. And so I knew that was not a starting place that was an option and I also was you know I was focusing on a younger population. My son was, I think 11 at the time, and some of the boys were even younger like seven or eight. So, I wanted something that was as easy to use as I could get it. And I had some background experience playing with Raspberry Pis, so I thought you could just put all this on a Raspberry Pi, have it boot up as part of plugging the device. Ideally get it to a point where somebody could just plug this thing in and they could plug in a microphone and headphones and start singing. That concept really stuck with me, but of course that implied a lot of other technical challenges to solve. So one was developing the software image for the Raspberry Pi and putting all the different pieces together. JackTrip is a tool that uses a lot of other pieces, in particular at the time it heavily relied on JACK. So having all of those things start up in the right order with the right settings and all these different things required essentially a coordination of different software. So I built an agent piece that ran on the Raspberry Pis that was part of the initial image that kind of coordinated everything. Then there also needed to be some way to control them like to tell them what to connect to. Like what server to get to, for example, what settings to use. So I started building out a basic cloud like microservice web interface to control the Raspberry Pis so that I could tell them connect to this JackTrip server, this port number and these settings, and disconnect. It kind of started out that simple, a very simple service in the back end and there was no security or anything it was just like as basic as it could be to do a proof of concept. So we took these Pis and we rolled them out to a small group, maybe about eight or nine boys. We scheduled the time to get together and we tried it out. And amazingly it worked for most of them, not everybody, but I wasn't expecting that, but it worked really well and they were able to sing for the first time in several months. That was awesome just kind of seeing their

faces light up as they past that threshold. So those were some of the initial challenges, kind of the coordination, the ease of use and from there it just kind of expanded out. We expanded it out to the rest of the chorus to, initially, the oldest boys chorus at [redacted], the young men's ensemble. Then other choirs in the area started hearing about it. So a few dozen of them started picking it up in the bay area and then it kind of went global. Hundreds, if not thousands of different organizations started picking it up worldwide and using it. As that kind of expanded and exploded it introduced need for all kinds of different things. If it's just [redacted] it's fine that there's no security really, but, if as soon as you have two choruses like you kind of really have to take that more seriously. So all these additional requirements really just started feeding into it, as it grew and became not just a proof of concept of “Is this possible?”, but as it became more of something that could apply to a broader audience out there on the internet. And so we just kind of scrambled to keep up with that as much as possible as it grew.

IR: Can can you tell me how the on and offboarding of developers happen?

IE3: Just to note it, my focus has more been on the JackTrip Labs or the Virtual Studio side of the whole JackTrip world and less so on the JackTrip open source MIT project that originated out of stanford about 20 some years ago. We've certainly worked on both and spent time on both. I've certainly focused on the Virtual Studio like that cloud ease of use user experience and then kind of the cloud service to help tie it all together. So for that, it was just myself for the beginning. For the first year it was just myself working on it and then after that I brought on a few people, three engineers, experienced engineers, and basically just spent a lot of time stepping through everything that I had built. One of the things we focused on was building test cases and better automated testing and I feel like that's a great way to learn as well as just it's good for the overall health of it. We've improved on the automation a lot, so we could really cut the cycle time down as much as possible and now it's at the point you just kind of click a merge button in GitHub and everything pushes out to either the test cluster or the production cluster. So getting all of those basic things in place, not only made us more efficient, but I think it helped bring people up to speed and what all the pieces were like how it worked, how it all fit together. That was largely just me spending a lot of time answer-

ing questions and stepping people through and them spending a lot of time like looking through the code and understanding it, how it worked and everything.

IR: What could be barriers for engagement?

IE3: For engagement of users or the engineers?

IR: Both, if you like. So we had the CLI problem for the users at first and maybe if we talk about the code itself or the codebase, or what else. Just to get into the project. It's like a follow-up question to the on- and offboarding thing. What could be a barrier, if somebody wants to engage in the project or what have your barriers been, if there were any?

IE3: It might be helpful to focus more on the open source side, because a lot of the Virtual Studio stuff is closed source and maybe that's not as applicable to the project working. But so the open source, looking at like my experience in terms of getting on board with it, also my experience bringing other engineers into that project I'd say one of the barriers is lack of technical documentation. Like this is the whole system, these are the different pieces of it, diagramming it out. “Here's how it works.” I don't think that exists, yet, probably should. And I imagine like part of it is just I know there's a general desire of the people in the community who work in the project to do an overhaul at some point and a lot of ideas and how that would look. Documenting an existing system when you plan to overhaul it is a barrier in itself maybe. I'd say that the Qt thing I've seen a lot of resistance on. It's a very popular framework. A lot of people like it, a lot of people use it. But at least the engineers that I've brought in are not as big fans of it and there's been some desire to use different frameworks and try out different frameworks. Also, I think a lot of the adoption of Qt stuff has been more primitive than it should be probably, or that it needs to be today. For example, threading, signals and slots, a number of other things like these are things that since 2000 have become part of the C++ standard and everything that's not part of the C++ standard is in Boost and I'm from more of a Boost background. I've always been a bigger fan of Boost and part of that's because it is standard track. So if you kind of adopt and you get used to something that's in Boost, it ends up becoming the standard and that happened to me for a lot of things. I did a lot of work in my previous startup in the 2007-2013 range heavily using Boost and

Boost threading, Boost ASIO, like a whole bunch of this stuff's now just part of C++11 or 14 and so I think it's a more natural track that if it's a primitive like that, if it's not a graphical thing, then it's better to use, in my opinion, something like boost. Or if it's already part of the standard, then you can go there just use a more modern compiler. So we've been talking for a while about that revamping that I know it's taking a few different threads. [redacted] has this project HackTrip, which reimagines a simpler version of JackTrip as something that could be more easily made into a library and used as a client. So it also takes out all the complexity of the server side of the project, because right now the project's really kind of two projects in one. It's like one executable that can act either as a peer-to-peer client. Actually it's three. It can act as a peer-to-peer client, as a client for a hub and spoke kind of server system, as well as the hub for a hub server system. Each of those has a bunch of code behind it that's different. He thought “Okay, what if, we just re-envision the hub client piece as an individual project.” Something that's a little bit cleaner as a library and that was a kind of a good starting point to look at what would a refactor look like for the codebase. I'd see the next step of that as trying to take out as much of the Qt stuff that's already part of a modern C++ standard. To me that would be a great next pass on that. And then probably rolling that refactor back into the larger project, re-envisioning what does the server look like, if it's built on top of HackTrip, for example.

IR: Okay. Do you think the problem domain has accounted for special approaches?

IE3: I'm sorry could you repeat that?

IR: Do you think the problem domain has accounted for special approaches like the monolithic structure of the project, of the executable.

IE3: I still don't quite follow.

IR: Just for example.

IE3: You mean the problem domain of online collaboration?

IR: The problem domain of the main functionality of the software for online collaborating music in realtime. Maybe there needs to be a special approach in the development organization or the organization of the code or something like this?

IE3: Yeah, I don't know. I mean you're saying that maybe this specific problem that we're trying to solve might rely a different organizational structure for how the code is put together?

IR: (Nods.)

IE3: I don't know. I haven't thought about that. I don't think so.

IR: Okay, how are artistic and research functionalities considered during development?

IE3: Artistic and research. I'd say the artistic side is thought through more from the perspective of what does the user experience beg like what is it looking for. If you're using this as a musician, what are your expectations for what functionality might be there, for example. Or for how other software in the music domain, music technology software, what are norms that are applied. How sliders work and how volume meters work and all these kind of things. So I think we take a lot of that into account of what is popular, what do popular DAWs do like how do you select audio interface inputs and outputs or even things like collaboration. Like Jitsy is, I was signing in here like how do I select my microphone and looking at examples out there is something we're frequently doing. I think that we frequently do it from the perspective of who is the audience that's using this tool. And another, I think, consideration that we take into account is what is the importance of different trade-offs of certain things. Audio quality is a big one. For example, with JackTrip it's really important to have high audio quality and arguably lossless audio quality. I know that even that can be a debate. I won't try to go into HD audio quality as that's a much harder argument, but I think with lossless you can get a much more general agreement. That when you're making music with people that it makes a big difference to be able to hear the sound as best as it's able to be reproduced. So that's a consideration, it's a requirement that's unique to this problem domain in a way. If you try to generalize it for example to collaboration software, then there's all kinds of trade-offs that are made in terms of quality of audio, where as long as you can kind of make out what somebody is saying, if they're speaking, then it's good enough. And that kind of a threshold doesn't apply at all to music, to musicians. So there are very unique considerations, I think, based on how it's being used. The latency is another big one. I think it's an obsession with minimizing is kind of how I think about it. There's always some

latency there and there's some latency that you can eliminate. I think it'd be very hard for you and I even if we're on JackTrip right now to sing happy birthday together across the planet. But minimizing it as much as possible so that people within a certain distance are able to do that is important and that's not really taken into account with other collaboration types of technologies for obvious reasons. What's interesting is, what we found is, although like a lot of these things we think, we've hypothesized are unique to music and musicians, we've really seen more and more indications that they're not unique to music, they're just generally applicable to human interaction. Even just talking with somebody in a conference like one-on-one even is very different on JackTrip than it is over Zoom, or the telephone, or something. That's an interesting learning for me. Something I wouldn't have expected is how different it feels to communicate with people in general even if you're not making music with them, when you have high quality low latency audio.

IR: Do you experience a change in the speed of the evolution compared to previous periods

and how does it affect your practice?

IE3: The evolution of the software?

IR: Of the software or the project in general.

IE3: I guess it is changing rapidly. For me I'm so in the weeds on a day-to-day basis that it doesn't feel like it's moving fast. Everything feels like it's moving far slower than I want it to. And whenever I look backwards in time and it's not just me, other people have said it's like look back three months, look back six months, look back a year, and then it's really obvious and I'm a little bit surprised at how much it's changed, because I just don't realize it while it's happening. I think the evolution of the broader project and by that I mean not just the open source, but that kind of combination of the open source with what we're doing at JackTrip Labs, I think that is really evolving rapidly. It's becoming much more useful and easier for people, for a much broader audience to use. That's cool. I don't think that there were that many people really using JackTrip in January of 2020. It was a pretty unique population base and now it's a brand. Now a lot of people know about this. It's won best of show awards or is at top conferences and stuff.

So, I think it's a lot more, it's getting out there and it's getting a lot more people using it. To me that's what matters to us like having the biggest impact is by making it accessible and available, and making people know about it and making it as easy for them to consume it as possible.

IR: Okay. I think the next questions might be a bit technical. If you don't want to answer them we can just skip them. One would be: Can you relate to making mistakes due to frequent changes whenever the system extends or adapts and explain your strategy to retain the overall picture?

IE3: Yeah, I think mistakes are common and normal and especially when you're iterating on a project quickly. We really do lean in towards pushing things out as quickly as possible and we also have a really small team and we don't have a QA organization. In Spunk we had a huge organization that purely was focused on essentially quality of the software and optimizing the efficiency and quality of the engineering teams and we don't have any of that, I think, in this context. It's an interesting switch for me personally, but I think whenever you're changing things often, you're gonna make mistakes, you're gonna break things. And I think we try to test things as much as possible, but it's also normal that there's going to be a lot of things that slip through. To me the only way to manage that is to respond quickly when you know that something's broken and we do keep track of that. We review all the bugs continuously throughout the week and we prioritize them. And to me bugs always take priority over new development, if it's a certain criticality level or if it's impacting a certain number of people. We always bump that to the top and try to fix it right away. Sometimes we'll fix it like same day, just push it right out. So I think responding quickly is important, but also looking at how did that bug make it in there, how did it happen. And in a way I try to do a little bit, maybe not a formal post-mortem, but in kind of a mental post-mortem on anything that comes up like that. And think of do we have the right checks in place. Like we have a release checklist that we go through regularly. There was a bug with the signing packages which made it through and we learned that the last few releases weren't signed properly and so we didn't just fix the signing we also said okay let's make an automated test check for that. Let's make sure that if the signing fails that our builds fail, let's also like add to the checklist to test the signing before we push something out. That's more of a

bigger thing that requires, I think, or that warrants a few extra checks. Some things are just thinking through. Some things you might address with a lighter touch, but I guess I always look for those opportunities to not just fix the issue, but to look at how we can avoid it in the future. If it were to happen again or there to be a similar situation or a similar bug could pop up again, how could we optimize that. And I think as you do that more and more over a longer span of time, you start to kind of build a much more resilient system overall to prevent things from creeping through.

IR: To which extent are time constraints influencing your work?

IE3: Time constraints is everything. Like everything is about time. Philosophically it's not just about our time it's like, it's about latency time. I think there is an infinite amount of work to do and so many things that we really want to do that would take years and years and years and the constraint is always how much time do we have. And in a way how much time, how much money? You can kind of cheat time a little bit with more money, but you'll always have some time constraints and money is just like one of those factors that kind of influences how fast you can go. So I am looking at everything and reprioritizing every day, because of time and recognizing that it takes time to develop things. It takes time to fix things, any kind of new feature that you take on like recently we added video. It's not just the time that it takes to develop it's the time that it takes to work through all the issues after you release it to a large audience and then it's the time that it takes to manage it over time and kind of maintain it and keep it up to date and keep adding features and so there's that. I'd say there's several dozen things that I feel like we should have them tomorrow and we can't. So like time comes into play because we can't have them all tomorrow so which one could we have sooner and that has the biggest impact. It feeds heavily into prioritization of everything that we're working on now and next and in the future. Ultimately, it shapes the project, it shapes like what it is today, shapes where it goes. It could be a different project, if we had more time. So I don't know, I'm getting too philosophical, but time it's the overriding factor for everything.

IR: Okay. When and why do you perceive time pressure during development?

IE3: When is always. Even before you start developing. When are you going to work on something or how much time is it going to take or when could it be finished. And then during time pressures come into play let's say there's a feature that's in progress and you know there's time pressures on what are the bugs, what are the customer support cases, like all kinds of time pressures that feed into that. And then priorities it's, sorry, I think it impacts everything during this cycle. It's who's working on what, how much time do they have, how much time is it going to take. I think things always take longer than you expect them to.

IR: Is the workload changing?

IE3: In terms of total amount of work?

IR: Yeah.

IE3: Not really. I think not practically, because you only have so much resource and the only way to change the resources is to have more people. I'd say that the workload that's queued is changing. You may have workload that changes to the most critical priority workload that's not being worked on. Let's say to me that changes depending on where you're at and how busy everybody is, but I think the total amount of productivity is fairly consistent as long as you're optimizing that productivity.

IR: How do you cope with it? With the changing workload?

IE3: We cope with it using, I'd say, standard planning techniques. We followed the scrum model so we have the sprint planning meetings, sprint reviews and we look at all the work that's coming up. We do these two week sprints and so we plan it out and make adjustments throughout. We have check-ins throughout the way if there's any adjustments that are necessary. Generally we try not to make adjustments because they should be the exception cases, if there's some critical issue that comes up or something. And I think that model really helps a lot to manage the workload, if you do the planning properly. How much available people are. If somebody's gonna be on vacation, you can take that into account. If there's holidays, take that into account. You can try to assess the amount of work that each of those tasks that you want to do is going to consume and then you basically just try to budget it in. Usually it works out like once you get to a cer-

tain point. It doesn't work out at all in the beginning. It takes three months or so for any team. And every time you change the team this clock resets, but any team kind of takes a few months to get into that groove and figure all that out. After that I think it generally works pretty well where you kind of know what you're going to get at the two weeks from now. I think trying to understand a plan or a roadmap beyond two weeks is overly optimistic. You can do it generally and you can have like a backload of things you want to do and prioritize all those backlog items. What I've learned is that they're always changing all the time. If they're not changing, then you're probably doing something wrong. Looking at the next two weeks and every two weeks is really the way to stay optimized and I think that works pretty well. It's hard for people to get into that mindset sometimes. Especially if they come from an organization that's more organized on a, I guess what's known as a waterfall model, or they're used to planning out long roadmaps or cycles or development processes or whatnot. But I've found that the scrum model works really well for our team right now.

IR: Cool. Have there been design decisions that rendered parts of the system hard to maintain or evolve from your perspective?

IE3: I don't know. I'm just trying to think as there's a lot of different pieces. One thing that's been difficult is, and I'm talking again of the broader thing, we originated out of this pandemic period like JackTrip Labs originated out of it and so a lot of the things that we built with the bridge devices and stuff originated out of the needs at that time and the industry is constantly changing. In technology it's like every few months you kind of have to reevaluate and rethink everything you're doing. One of the things that changed over the last two years is the audio interface efficiency. I started out thinking that you needed to have specialized audio hardware for this to work because all these USB interfaces and microphones and everything that I tested had 20 milliseconds themselves, just on the interface and there's no way you could make music work with that. So I started out with that model of the Raspberry Pis and I think since then it's changed. You can buy USB devices today that have much lower latency like sub seven at least, but generally you can get good ones sub five. And if you're willing to spend the money, you can get really good sub one millisecond interfaces that are USB 4 Thunderbolt 3 and it's just like recognizing that five years from now everything's going to be sub one

millisecond. It's just the way things go. So that trend really stood out as something for me and if I were to start today, I would not build bridge devices.

IR: Okay.

IE3: And the complexity of managing those devices. And whatnot, I think all of that is no longer necessary. We've made, I'd say, that this kind of really sunk in about a year ago for us, maybe a year and a half ago. And about a year ago we started heavily investing into the software app as a primary means of connecting to Virtual Studio and that led to like a lot of investment ultimately back into the open source project. Like we didn't want to create a separate app we wanted to just feed it in and contribute it back to the community. So I think it's good all around, because that means that the engineers that we have at JackTrip Labs are now on a daily basis working on that open source project and digging into it and adding to it and making it better and fixing bugs. So it's making it better for everybody. That eliminates the whole complexity of having to use separate hardware devices out there. So I think we're kind of going through that transition really over the past year. We've been going through that transition and by the summer we released a new version of JackTrip that was more integrated with Virtual Studio. That was a big turning point of being able to really start to phase out the bridge devices and then by actually just last month we stopped selling them. Just pulled them off all together from our website and everything. So that was kind of the last in that, not the last step, it was the next step in that transition. I think the last step will be finally no longer supporting them and then at that point we'll be able to just remove all the stuff that's specific to the bridge devices.

IR: When discussing parts of the system are cohesion and coupling considered for creating boundaries and connections according to design principles?

IE3: I'm sorry what is the question?

IR: So the question is what about cohesion and coupling?

IE3: You mean it's a good thing or a bad thing?

IR: Okay it's a very technical question, sorry. It's about how the parts of the source code are organized with regard to how cohesive they are. So you want to have, for example, a

class with several methods and the methods refer to each other so there's a reason they are in one class and not splattered over several classes that's cohesion. And coupling is the other thing you have several classes that have to use, for example, methods of each other. And an aim would be to have loose coupling so you don't have to change a class that is connected to another class when you're doing maintenance work or adaptions or what else.

IE3: (Nods.) I think different languages treat these things differently. On the studio side we had that opportunity to start from scratch and to build out modern microservices that kind of follow best practices and principles on that. On the open source side I don't know it as well as other people. I've haven't done a bunch of changes in there, but I know there's a lot of coordination across classes using signals and slots. In some ways I think it's good, in some ways I think it's unnecessary that things are somewhat coupled together through signals that don't necessarily need to be like or shouldn't be as coupled as they are. And I think it makes it hard to understand how those things work together in some senses, but again I'm not really enough of an authority. I haven't spent as much time in the source code as other people on the open source project.

IR: Okay. One last question. I think we have been to that, but maybe you can elaborate a little bit more. Imagine what would be different, if the project started today.

IE3: I think, if it started today, it would use like modern C++ libraries, C++14 and a lot less of the Qt stuff. I think it would have a cleaner distinction between what is the library that could be reused by different implementations of JackTrip and the various end applications. So, for example, I was talking earlier about how JackTrip today is kind of four things: the peer-to-peer, the hub client, the hub server and I think there could be a library that's just kind of reused across those and possibly multiple libraries that provide more common stuff to them. I think that would kind of make it a bit easier to do different things and be a little bit cleaner. I think it'd be better documented. And especially just the act of creating a library kind of forces you to document what the library is providing and how to use it and that makes it I think more reusable beyond the current applications that are leveraging it.

IR: Okay, wonderful. Thanks, thank you very much. I will stop the recording.

12.1.4 Interview #4

Date: 2023-02-02

Participants: Interviewer (IR), Interviewee 4 (IE4)

IR: Okay, so the software architecture of the system is like a bare brick work in construction that carries the source code. But during implementation, trade-offs require deviations from such a blueprint. These are just natural and can't be avoided, however, if the planned architecture is violated, the system becomes hard to maintain and evolve until it probably breaks. Imagine a bungalow with several unplanned extensions on the roof, leading to cracks in the building. In software engineering, that indicates architecture erosion. And I'd like to learn about the JackTrip software project, its origin, evolution, and environment to find possible reasons of that phenomenon. Cool. Okay, so the first question would be how long and for what motivation are you contributing to JackTrip?

IE4: So I guess my first contribution came at the start of the pandemic. So I'd used JackTrip before a number of times for telematic performances with various projects. There's a new music ensemble that I play with called [redacted], and I've done some telematic stuff with them before, as well as with some of the university students at Monash. But of course, with the changed environment of the pandemic and not being able to be in person or on campus, obviously there was a renewed interest in it. And so I got on board with, well, there are a few things that I wanted to do with it to begin with. One was I only ever used it point-to-point before. And so I was thinking, if I could run multiple instances of it at once and script things, then we could make sort of like effectively a hub server and connect multiple clients to that. So I was starting to look at being able to do that only to find out that was a feature that I think had just appeared in version 1.1 or something like that at the time. So it's like, well, that was an easy job. My work here is done. But then there were a few minor bugs that came up with the current implementation or the then implementation of the hub server mode. So for example, if a client dropped out, as it inevitably would if you were leaving a session, the zero sound flag wasn't being honored. So you'd get this buzzing sound that was just the repetition of the final packets that they'd sent through, which is particularly annoying if you're trying to run any sort of ensemble thing. And so I thought that's probably a fairly easy fix in

the code. And so I started to acquaint myself with the code and sure enough it was easily enough solved. And so that was sort of the first thing I did and then started to – There were a few issues with, I guess, the way name resolution was happening on a dual IPv4, IPv6 network. So I fixed some of those and then thought, well, why not add IPv6 support properly? So I started to do a lot of work improving the network code, then got in contact with the Stanford crew just to let them know that I'd been developing the code. I had these patches and if they wanted that they were more than welcome to it. And they embraced it and brought me on board as one of the sort of open source dev team, which was great because I think at that stage they just moved back to GitHub from another system and I just sort of started becoming involved with the project that way. Then I gradually sort of did a lot of work on some of the behind the scenes code. So there was a lot of synchronous code that I got rid of so that improved how the networking was running. There was a very interesting way in which sort of multiple sockets were bound to the same port. One half closed, which was something I'd never seen work anywhere else and it worked as obviously an edge case in IPv4, but didn't work in IPv6. So I had to do a lot of fixing of things in the backend there. So it was mostly behind the scenes stuff. The main thing that I added beyond then was a graphical user interface, because I had a lot of staff and students at the university who were really interested in the project and interested in using it until you started talking about the command line and then you'd see their eyes glaze over and at that point you'd invariably know that you'd lost them. So that kind of became my main focus then. And I'm still slightly involved in coding stuff at the moment, but I've just been on a UK tour with the new music ensemble. I've got a few major conducting gigs coming up and some major compositions that I need to get through and some commissions that I need to finish. So coding has kind of taken a bit of a backseat at the moment, but hopefully it's something that I'll continue to just make contributions to as I get the chance.

IR: Okay, please tell me about challenges in the project, the organization, processes, tooling, but also with technology adoption and infrastructure for example.

IE4: In terms of the coding side of things or just in general, or?

IR: Both.

IE4: I guess one of the big issues that we constantly run into is that obviously it was a protocol that was developed a long time ago in a much simpler era. I think the first version was like 2000, like maybe mid to late 2000s. So it's been under development for a long time. Obviously design decisions that were made back in the day worked then, but then trying to add extensions and new features is a tricky thing to work around without breaking compatibility. So for example, one of the things that I added was the ability to send an exit code when you disconnect from a server so that it knows that you've actually quit intentionally and it's not just the connections dropped and it's waiting for packets. So kind of having to shoehorn that sort of stuff into the protocol was a bit tricky, especially since each audio packet has a very tiny header attached to it, which makes sense because you don't want to have too much data there. You want it to be fast and responsive and not sending a lot of additional information with each audio packet, which can be quite small if you've got a low sort of frames per period setting in JACK. So it made sense, but then of course it's hard to add control commands and things like that within the protocol. The other one that I ran into was in terms of adding authentication to it and there's two different parts that have been followed with that, because of course there's the open source standard JackTrip and now there's the JackTrip Labs version of things. Of course their authentication runs through their own OAuth servers and things like that, whereas I added an actual mechanism for authentication into the protocol itself and once again that required a lot of creative ways of getting around the fact that the initial TCP handshake only sent a 16-bit port number. Thankfully it sent a 16-bit port number within a 32-bit integer so there were 16 unused bits which I could hijack. But it's been a matter of trying to work around those limitations in a way that hasn't broken backwards compatibility, but that's sort of maintained the protocol. But I think there's always various talks at developer meetings about how we can get around those things and who knows, eventually there may be like a JackTrip 2.0. We just decide to re-engineer things, but I think that would be a big project and at the moment I think everyone else is focused on other things.

IR: How about on- and offboarding of developers?

IE4: Yeah, I think that's been probably not too big an issue. Once again I think that the main split now is that there's sort of the open source developers and there's really only a

handful of those who are kind of active at the moment, but all of them have various things that they're focused on that don't necessarily have too much overlap. So there are various people taking care of various bits of the project and then of course there's the JackTrip Labs crew. There's some developers there on payroll who are purely contributing to this project. I mean one of the good things about it is that it's stayed fairly modular in how it's been coded. I think, ideally one of the things that's been talked about, but which would be a lot of work, would be taking out the core component of JackTrip and creating a library that then any other software package could use as they see fit. So you could have a core JackTrip library that other audio applications use and they can take advantage of it however they want. Obviously that's a lot of work. I think some people have started various drafts of it, but it hasn't necessarily gotten very far, at least in the way that it's coded at the moment. So for example, when I added the initial graphical user interface, it's the same program as the command line program. You can run it from the command line with a whole lot of command switches and it'll run in the old fashion that it used to in command line mode. If you run it from the command line with no command line switches, so no arguments, it'll launch the graphical user interface and that's typically what happens if you click on the icon in Windows or on the Mac. And the good thing is when you build the application you can still choose to build the command line only version. So if you're running a JackTrip server somewhere and you're obviously not going to need the user interface you can build this pared down version that only has a command line and you're not creating a bigger package than you need to, you're not wasting resources on that and you don't have to have these additional libraries installed. The same thing then happens with the JackTrip Labs version which is the kind of commercial version of the app. Once again you can still have the app and you can choose to launch into the conventional graphical user interface or JackTrip Labs interface, which then uses their servers and which are for the most part a paid for experience or you can just run it from the command line. So all of the software components that make those things happen obviously use the same core JackTrip code, but they're quite independent as modules. Well, for the most part at least. If something changes within for example the JackTrip Labs parts of the code, it doesn't affect the standard graphical user interface or it doesn't affect the command line version. So I think that's the way that the

architecture currently sits. It's kind of insulated us from maybe some of those onboarding issues that might have occurred, if it was a more monolithic piece of software. So, as I said in this way the JackTrip Labs people can tinker around with their part of the code and not fundamentally damage things for what everyone else is doing, which is currently one of the strengths of how it's structured.

IR: What could be barriers for engagement?

IE4: Yeah, in the coding front or in the user front or?

IR: In the dev side.

IE4: I think it's the same standard barriers of engagement that exist for any software project. Especially, now that it does have the commercial side of things behind it, which means that the code base is automatically bigger and there's a lot more code to come to grips with. Now if you're just planning on looking specifically at the open source side of things, there's less to get your head around. But you still have to work out which parts of the code are important to what you want to do with the project. So I guess now that it's a bigger code base maybe one of the barriers to entry is just getting accustomed to the code. But also it's the same sort of thing that exists for all sorts of projects where there's going to be an initial learning curve no matter what you get involved with and it's always going to take time to sort of come to grips with those things. And that can probably be kind of daunting for some coders especially if they don't really have any connections to any of the current open source coders. I'd like to think that for the most part we're a fairly approachable bunch and people can ask questions and there's obviously the discussion board and the issues board on GitHub. But even then for someone just coming in from the outside that can still be kind of a daunting sort of thing to come into. But hopefully, I mean generally when people raise issues or raise discussions there's usually been some sort of engagement from the current devs. So I think we tend to stay on top of that sort of stuff usually so that if people do want to engage with it, they at least have those answers. I imagine there's generally far more people who are interested in using it than in developing so thankfully there's a separate Google group that takes care of a lot of those sorts of questions. The main barrier to entry is just being able to try

and work out what's happening with the code and how it's structured. And what are the key important parts of it that you'd need to alter to achieve your goals.

IR: Okay, how are artistic and research functionalities considered during development?

IE4: I guess in terms of the open source coders it really depends on who's programming what. So I think probably on the open source side there's maybe a little bit more of a consideration of the academic side of things. A lot of especially the contributions that [redacted] makes are geared towards I guess what he's doing in his artistic practice at the time. I mean there are some historic ones that really haven't even been, they're not necessarily that useful to the public. They're very specialized modes that exist within the code for specific projects. And they've led to other generalizations that have helped others, but some of them have been just very, very specific academic project driven code. I guess for my part the addition of the authentication stuff for instance was driven by the desire to use the software in a university environment where IT generally are not so enamored with the idea of having software that's not password protected and that anyone could potentially log into a server for. I think they sort of became very averse to that sort of thing after initial Zoom bombing type things at the start of the pandemic. So having that sort of code in place was important to help our use case with the university and getting a server set up. Basically all of the stuff that I've been working on, I mean the networking improvements not so much as they were just general things, but the graphical user interface and that sort of stuff, was really driven by my own practice and my own desire to see the software embraced amongst the student population who are not familiar with the command line and by musicians who are really not comfortable with the command line at all. So I guess in that sense, a lot of what I personally do is driven by what I want to see JackTrip do in the university environment and within my own research settings and with the idea that hopefully then that has flow on benefits to the general community. But once again I can only speak for the open source side of things. On the commercial side of things I think really they're more interested in, – and understandably so – in the overall user experience and in creating a user interface that really fits with the server functions that they're offering as part of their product. Which totally makes sense for what they're doing.

IR: Do you think the problem domain itself has accounted for special approaches in the design of the software, for example?

IE4: Yeah, I would have to say yes. And part of that is in how it's leveraged other existing software tools to do what it needs to do. I guess the name JackTrip highlights the fact that it was always built initially on JACK. There are developers who are working on other audio interfaces for it and trying to get away from that dependency on JACK. But originally using JACK was kind of almost by necessity in terms of having a very low latency audio API that could get data through the system as quickly as possible. So of course the biggest amount of latency is going to come from the internet connection and there's not a lot you can do to minimize that. The connection is what it is. So the less latency you can get within the rest of the system, the quicker you can get the audio from the buffers out to the sound system, the more you can hopefully push latency down to a value where you're going to be able to play in real time with someone else. I think the desire to minimize latency has really sort of governed a lot of the initial decisions. Once again I haven't tested at all what the latency is like with for example the RtAudio API. I'm sure it's probably fairly comparable to JACK, but I think there's maybe now a push to sort of not worry so much about those low latency technicalities as there is to make it user-friendly for users. Especially on platforms like Windows where there are a lot of conflicting audio APIs, a lot of legacy APIs that exist and all sorts of peculiarities with its sound system. Initially the decision to use JACK was a very deliberate one and it's something that I love because most of my work is on Linux or on Mac and Linux and I'm currently running JACK on the system that I'm talking through at the moment. It organizes all the sound of my computer. So that's not something that I've seen as an issue, but for others having to rely on that, installed as the third-party piece of software, it becomes a liability and so there's a bit of a push at the moment, especially on the JackTrip Lab side of things, but also from some of the open source developers, to get away from that dependency. Just so that less technically inclined users can have access to things. I think a lot of the initial decisions about the project were based heavily around minimizing latency at the cost of everything else and that's where some of those other bottlenecks that I've had to get around have come into play. And now there's a bit more of a focus on things being user-friendly that sometimes clash with the low latency

ideals, but I think there's been a lot of work to kind of find the best compromise between the two as much as possible.

IR: Do you experience a change in the speed of evolution compared to previous periods and how does it affect your practice?

IE4: Certainly, as I kind of got on board at the start of the pandemic. So already that was a time where JackTrip went from a fairly sleepy sort of research project to being thrust into the limelight. You look at the GitHub commit records and it's fairly sedate for a long amount of time and then suddenly 2020 hits and all of a sudden there's all this code being committed. I think once again that sort of got kicked up a notch when JackTrip Labs not only became a thing, but once they started sort of hiring staff to do full-time coding for it. So I think at times that's been a bit tricky, because there's been so many things committed that branches have gotten out of whack quite easily at times and sometimes I think people have been a little bit trigger-happy on committing code that maybe wasn't quite ready for prime time. But I think we've gotten to a point where we've sort of ironed out a lot of the processes involved. It really came to a head with a release that really broke something bad there – I think on one of the platforms I can't remember which off the top of my head. Basically they had to launch another release pretty much immediately after it so it's ideally a situation that should never have happened, but I think that led to a lot of discussion about how we can make sure that doesn't happen again and what sort of processes we need to engage to make sure that the code gets into the code base in a timely fashion, but without breaking things.

IR: So how is your practice affected?

IE4: I guess it's not too badly impacted at the moment. There was a time where especially I did some initial work on the JackTrip Lab side of things and there was a time where people were just committing stuff really quickly to a point where it's like, well that's great that people are making these contributions. But suddenly now I have all this code here that is out of date and I need to refactor. Now I'm mostly more involved, almost exclusively involved, in the open source side of things and I think that's a slightly easier target to code for. There aren't as many active developers working purely on that side of things. So having that split where there's a clean delineation between the open

source side of things and the commercial side of things has been really helpful in just making sure that my practice kind of continues on as-is and it hasn't really been a major issue at all.

IR: So your contributions are more in a hobby way or for academic purposes or?

IE4: For academic purposes. Although that said, just because of my workload at the moment, its' almost feeling more like a hobby at this point in time. But it's yeah definitely more on the academic focus side of things. So thinking what's useful for the app in my own practice as a musician and as an ensemble member and within the institution that I work. So I guess that's mostly where my focus lies when I do have time to program it and of course being at a university where I have access to cloud services and being able to set up my own servers, I just I haven't had the need to use, for example, the JackTrip Labs service at all. The major split between the open source side of things and the JackTrip Labs side of things is that the commercial side of things really takes away the burden for people of running their own servers and having to know about the networking side of things. How port forwarding works, all that kind of stuff. So it takes away that burden. But when you're working in an academic space where you have institutional resources, where you can run your own servers and where you're getting students to maybe try creating their own point-to-point networks and working out the networking side of things themselves, there's not really that need for third-party commercial servers. So that's pretty much why most of my efforts are focused on the open source side of things. I think also there's still a very healthy community for not just the non-academic world, but the world of practicing musicians. Musicians have been using this software for a while. Especially those who've been using it since before JackTrip Labs was really a thing. So they're already used to running their own servers, they know how to do all that sort of stuff, they've already got an investment in their own infrastructure, in their own equipment, and they're comfortable at this stage doing that. Most of my work is geared towards, as I said, the academic community that I'm a part of and also I guess with flow on effects to those other users who are comfortable running their own servers and running their own systems without relying on paid servers.

IR: Can you relate to making mistakes due to frequent changes whenever the system extends and explain your strategy to retain an overall picture?

IE4: Yeah.

IR: We can split the question up, if you like.

IE4: Once again the good thing about the way things are at the moment is that there is that careful delineation between the commercial and open source side of things. When I did have a foot in both worlds of that it was a lot harder to keep track of the pace of things that were happening on the commercial side of things. And I guess in terms of strategies for staying up to date with things it would generally just be having to merge whatever had happened elsewhere back into what I was doing seeing if there were any clashes and then working around that just as quickly as possible, and trying to do that as often as possible so that it didn't get out of hand. Even in the early open source days where there wasn't as much happening, people had their own separate branches that they were working on more often then, so there weren't as many commits to the main dev branch as there are these days. So things would often splinter out in quite radically different ways. That was always kind of problematic where there were long times between commits to dev and things could get radically out between individual contributions. I think as we've gone along one of the things that helped is that people have been committing to dev more often. There've been more frequent pull requests. So there's been less chance for things to get drastically out. You can very quickly and easily see okay there's been a few pull requests I'd better merge this back into my code. And if you do need to fix clashes, usually they're fairly small. Whereas in the old days, before there was quite as much commercial interest and we had to work out how to sustain things between the commercial and open source side of things, there was much more scope for that to happen where you'd have to really do a lot of work to try and work out where things had gone, had branched off differently, and what needed to be fixed for a merge to be able to happen. So I think it hasn't been too big an issue, recently, thankfully. Obviously mistakes do still happen, but I think not at the frequencies they used to. When you have huge clashes in the code then you'd have to manually sync things. And occasionally within that process of getting things back into sync sometimes things would go

horribly wrong. Often when they did it didn't take long to track things down and fix it beyond that, but it was still a bit of a hairy process at times. I hate to speak too soon because who knows how things will go, but hopefully those sorts of major merge issues are a thing of the past. I think just process wise we tend to, we've gotten on top of that sort of stuff, hopefully.

IR: To which extent are time constraints influencing your development work?

IE4: Hugely. It was definitely much easier back in the pandemic. Well, I say it was easier back in the pandemic. You see all those videos of people who are like making sourdough and all that sort of stuff and I was like I don't know how people have time, because university just kept going. And not only did it keep going, everything had to be pivoted online all of a sudden. When you're trying to do music online of course that's just a nightmare to deal with. So there was a lot of stuff that had to happen really quickly and it was a bit of a crazy time. But still throughout that, because there were no outside gigs happening, there wasn't as much playing happening, I was still doing quite a bit of recording, but not a huge amount and they were fairly small and manageable recording projects. Whereas these days, as I was saying, I've had and have coming up some major conducting commitments, quite a few commissions to get through and really need to start getting on top of my PhD project itself more, which I mean JackTrip kind of is now tangentially involved in. But it's a matter of really getting on top of all that sort of stuff. So I think these days time constraints are a huge factor in terms of my development time. I just haven't had as much time as I would like, but as I was saying thankfully now that things seem to be running quite smoothly in terms of how pull requests happen and how the architecture's working, there's some work that I was doing on adding a user interface for the authentication, for the inbuilt authentication mechanisms. It's stalled, because I just haven't had time to look at it. But once again I don't feel like I have to do a lot of work to change what I've already done, because as I said the way that the mergers are working now is much smoother than it was previously. So it feels like there were times before where I was quite busy and had limited time and it would be super frustrating, because I have to do all this extra work to get things that I'd fixed back into the now updated code. Whereas I don't feel quite that much pressure

now, which is good even though I do have a lot less time to sort of devote to the project at the moment.

IR: So when in the development stage do you perceive time pressure or did you perceive time pressure?

IE4: Oh, just in terms of my own time that I can spend coding at the moment. My time is sort of diverted elsewhere at the moment. So it's just a personal lack of time to spend on everything.

IR: Okay. Is the workload changing?

IE4: Not so much. I mean once again as a result of the things going back to kind of normal post pandemic the level of playing commitments and conducting commitments that I have are increasing and as I said I'm starting to get a lot of commissions coming at the moment. So that sort of stuff is eating up a lot of time that previously I might have spent on coding. It's just trying to find space in my own research workload to actually code JackTrip stuff. Especially when I have other PhD related coding that I need to do on another project that kind of is taking a little bit of priority at the moment.

IR: So you're contributing to other open source software related to your research?

IE4: There's an animated graphic notation software called the Decibel ScorePlayer that I've been developing as a member of the [redacted]. We created it within the ensemble. I've been pretty much the sole developer for a long time. It's currently open source, but it's still just my contributions at the moment. So that's kind of the other project that I'm contributing to. As I was saying technically open source, but really just my own personal project within my own research and within the work that the ensemble is doing.

IR: Okay. Have there been design decisions that rendered parts of the system hard to maintain or evolve from your perspective?

IE4: Once again I think really it's just some of those bottlenecks that existed in terms of there not being a lot of header space in individual packets, and not being a lot of ways to get around sort of the limitations of the protocol, and having to find creative ways to get around the limitations of the protocol, but that's probably been the main design decision

that maybe made things a little bit hard to maintain. A lot of the code initially in terms of how the networking was first set up, the way it was being handled did make things a bit difficult to maintain. It certainly made implementing IPv6 support more work than it seemed like it should have been. So the underlying architecture just didn't work with it. As I said a single socket was originally being shared by two parts of the program. Never seen it done that way in any other software seemed like quite an ingenious workaround. That worked for IPv4, but didn't really seem to be part of the socket specifications so kind of was maybe a loophole that was exploited and that then, in the way that IPv6 sockets were coded, didn't quite work out. So there was a bit of maneuvering around that and there was a bit of maneuvering as well in terms of getting rid of some synchronous blocks of code. We had some odd bugs where the code would block and wouldn't unblock for various reasons in these old synchronous code blocks and getting everything working asynchronously sort of just solved all those issues. So in terms of the way the original code was written there are large chunks of it that in terms of fixing the networking stuff, in terms of fixing some of the hub server stuff, that I just replaced with newer code, that hopefully will prove to be more maintainable going forward while still preserving the exact same behavior. Well, at least in terms of the networking protocol of the old code. So hopefully some of those roadblocks have been gotten around in the past few years. Obviously limitations in the protocol itself and what's being sent is the big one that we really can't address without either a lot of creative solutions or breaking backwards compatibility. So I think that would probably be the big thing to discuss going forward. Although there have been discussions about ways that we can get around it like when we're dealing with hub client and server mode keeping the initial TCP connection open as a means of then sending OSC control messages back to the server. And so that sort of thing has been floated a lot and would definitely be a possibility going forward. But once again I think that's the sort of thing where we would need to get a lot of the stakeholders in the room together to talk about what's the best way to define an address space to do things here. So you don't want to create this whole new command system that then has all of these pitfalls and problems that we make for ourselves later. So it's one of those things where it's like “Let's get this right the first time”. I think we haven't sort of made any real progress on that at the moment.

IR: When discussing parts of the system are cohesion and coupling considered to for creating boundaries and connections according to design principles?

IE4: Maybe not so much. I mean the big issue for me is I haven't had a chance, especially because of the time zone, to make it to some of the dev meetings for quite some time actually. So I can't really speak to much of what's happened in recent dev meetings. I know there have been, when we have had dev meetings in the past we have talked about all of the architectural choices that we're trying to bring on board and that sort of thing, but maybe at times there's not as much discussion as we potentially should have. But so far it's mostly worked out in the long run. But as I was saying I can't really speak to what's been happening recently in the dev meetings. I think having those regular sorts of meetings has been useful for that sort of planning, but unfortunately time constraints wise I haven't been involved in a lot of that of late. Just from what I've seen in terms of the conversations elsewhere tangentially to that I suspect that maybe some of those discussions have been a bit more formal of late which is good. But once again don't quote me on that, because I haven't been there and I can't really testify to that.

IR: Okay we almost reached the end. The last kind of question would be: Imagine what would be different if the project started today.

IE4: I guess it depends once again on the exact nature of how it started. Because it could potentially still just start off as a sort of once again a niche research project that then has ballooned out in the way that it has. I suspect were it started in the age of the pandemic where there was that much more interest around it, it might have started as more of I guess an open conversation and as more of a sort of communal thing. In a way you almost don't need to imagine if it started today because there have been projects like SonoBus that were started in the pandemic. Admittedly they had slightly different design goals. And I think SonoBus has always been more prepared to prioritize sound quality over latency. That's the same with some of these other projects that have existed. I think Audiomovers is a similar one. It's really about sound quality rather than minimizing latency so they do have slightly different aims, but for something like SonoBus that was something that sort of emerged, I wouldn't say almost overnight, but sort of really rapidly. Because of it being developed in an entirely different era, an entirely different

time to the initial JackTrip work. So SonoBus, for example, was able to leverage things like the JUICE toolkit. It was able to leverage things like audio over OSC and all of these other things. Once again I'm not sure, I haven't looked into audio over OSC enough to know how it fares as a protocol and how efficient it is as a protocol. I would imagine it would have to be less efficient than what JackTrip is doing where it's just for the most part sending raw audio data with very minimal header information. But the trade-off for that slightly lower efficiency is probably much greater flexibility. And it's also when you look at when JackTrip was first being developed and what internet connection conditions were like for a lot of people. People generally didn't have access to the same broadband connections quite so readily as they do these days. So that level of efficiency in this era probably doesn't matter so much. You can get away with sending more network data over a high bandwidth link and gain that flexibility at the the cost of efficiency. And once again I said that was built mainly using existing toolkits. JackTrip really is kind of created largely from scratch. The main frameworks that it uses are just the Qt frameworks, which of course are not particularly geared towards audio or anything like that. There's no ready-made audio toolkit that it was using other than of course JACK. That's the big thing that it was leveraging there. This is one of the things I think the commercial side of JackTrip is looking to improve upon. As I mentioned earlier, in getting rid of that dependency on JACK and allowing for a larger variety of audio toolkits to be used whereas as something like SonoBus has just had ready access to that sort of flexibility right from the get-go simply because of the era in which it was coded. I think speculation is always hard, but I imagine that's probably one of the main ways in which it would have changed, if it was developed in this era, in that it would probably have leveraged and probably had the opportunity to leverage existing libraries to a much greater extent. Arguably that does put you then at the mercy of third-party libraries and for a project like JackTrip where it was all about latency having that raw access to JACK was a really important feature of the code. But once again is it as important in the modern software landscape? Possibly not. So I think that's probably the main difference that we might have seen. That and also there probably would have been more people interested in the space and so there might have been greater number of collaborators from the get-go, but then once again you could look at SonoBus as kind of an ex-

ample of the opposite of that where one person was able to sort of piece together these existing frameworks into something quite rapidly. Although once that was released it probably now has quite a lot of contributors behind it. It's probably no longer a one-man job. Once again I would have to to look into what their GitHub is currently like to know how much contribution was coming from others, but I think that probably covers most of how it would have been different.

IR: Okay. Cool. Thank you very much.

13 Declaration on Oath

I, Sven Thielen, hereby affirm in lieu of oath that the master thesis at hand is my own written work and that I have used no other sources or aids other than those indicated. All passages quoted from publications or paraphrased from these sources are properly cited and attributed.

The thesis has not been submitted in the same or a substantially similar version, not even partially, to another examination board and has not been published elsewhere.

Düsseldorf, May 23rd, 2023 (Sven Thielen)