Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Language Technology

MASTERTHESIS

Incorporating Research Software in Research Data Management Infrastructure: Architectural Design for Sustainable Scientific Software Integration

Hagen Peukert

Field of Study: Computer Science
Matriculation No.: 6417586
1st Examiner: Prof. Dr. Chris Biemann, Universität Hamburg
2nd Examiner: Dr. Katrin Stierand-Schöning, Universität Hamburg

Language Technology
Department of Informatics
Faculty of Mathematics, Informatics and Natural Sciences

Universität Hamburg
Hamburg, Germany

A thesis submitted for the degree of

Master of Science (M. Sc.)

October, 2026

Incorporating Research Software in Research Data Management Infrastructure: Architectural Design for Sustainable Scientific Software Integration

Masters' Thesis submitted by: Hagen Peukert

Date of Submission: 3-31-2026

Supervisor(s):
Dr. Katrin Stierand-Schöning, Universität Hamburg

Committee:
1st Examiner: Prof. Dr. Chris Biemann, Universität Hamburg
2nd Examiner: Dr. Katrin Stierand-Schöning, Universität Hamburg

Universität Hamburg, Hamburg, Germany
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

Language Technology

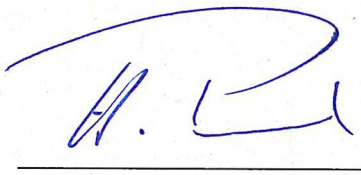
Affidavit

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe. Sofern im Zuge der Erstellung der vorliegenden Abschlussarbeit generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten Wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

I hereby declare in lieu of an oath that I have written this thesis for the Master's degree program in Computer Science independently and have not used any aids other than those specified – in particular no Internet sources not named in the list of sources. All passages taken verbatim or in spirit from publications are labeled as such. I further certify that I have not previously submitted the thesis in another examination procedure. If generative artificial intelligence (gAI)-based electronic tools were used in the course of writing this thesis, I affirm that my own work was the primary focus and that complete documentation of all tools used is available in accordance with good scientific practice. I take responsibility for any incorrect or distorted content generated by the gAI, incorrect references, violations of data protection and copyright laws, or plagiarism.

Hamburg, 16.3.26

Date



Signature

(Hagen Peukert)

Abstract

This paper presents the design of a software architecture for integrating heterogeneous research software into an established research data infrastructure at a university institution. The research software can be understood as an autonomous application in the sense of a micro-service architecture, which is integrated via an interface at the data layer but otherwise operated, maintained, or extended independently. The prerequisite for such integration is, on the one hand, that the research software keeps its data accessible and, on the other hand, that the research infrastructure, in which the architecture is to be implemented, establishes the appropriate research data management standards. This includes, in particular, a scalable data repository. The software architecture, which is itself designed as a modular monolith, takes over some of the tasks of the services on the presentation layer (user interface) and the data layer. These include the management of interfaces, their modification and adaptation, the transfer of data to sustainable storage, and the homogenization of data as far as possible according to the standards of the respective research discipline.

Using the *Attribute-Driven Design (ADD)* method, the architecture is developed in three cycles: development of a basic structure, integration of functional requirements, and integration of quality attributes. For each cycle, the design purpose, the choice of architectural elements, design decisions, responsibilities, interfaces, analyses, and diagrams are provided with explicit justification. The evaluation is based on the *Lightweight Architecture Evaluation*, a shortened version of the *Architecture Tradeoff Analysis Method*. The evaluation is not part of the original ADD method, but could be included in the method as a feedback loop.

To develop the architecture, the requirements are determined by a field and a needs analysis. The field analysis is based on the evaluation of national and international surveys, conference reports, case studies, and descriptions of best practices from publicly funded infrastructure projects. Part of the needs analysis can be inferred from the environment, while more specific needs are obtained from individual research projects and prototyping. With the help of requirements management, these findings are translated into quality attributes and functional requirements. The entirety of the analyses' results, together with the findings from the state of research and the theoretical background, form the knowledge of the domain.

Zusammenfassung

Die vorliegende Arbeit enthält den Entwurf einer Softwarearchitektur für die Integration heterogener Forschungssoftware in eine etablierte Forschungsdateninfrastruktur einer universitären Bildungseinrichtung. Dabei kann die Forschungssoftware als autonome Anwendung im Sinne einer Micro-Service-Architektur aufgefasst werden, die über eine Schnittstelle auf der Datenschicht integriert, aber ansonsten unabhängig betrieben, gewartet oder erweitert wird. Voraussetzung einer Integration ist, dass einerseits die Forschungssoftware ihre Daten zugänglich hält und andererseits die Infrastruktur, in die die Architektur implemtiert werden soll, selbst über entsprechende Standards des Forschungsdatenmanagements verfügt. Dazu zählt insbesondere ein skalierbares Datenrepositorium. Die Softwarearchitektur, welche selbst als ein modularer Monolith konstruiert ist, übernimmt einige der Aufgaben der Services auf Präsentationsschicht (Nutzerschnittstelle) und der Datenschicht. Dazu gehören das Management der Schnittstellen, ihre Veränderung und Anpassung, die Überführung der Daten in nachhaltige Speicher und die Homogenisierung der Daten nach den Standards der jeweiligen Forschungsdisziplin.

Mittels *Attribute-Driven Design (ADD)* erfolgt die Ausarbeitung der Architektur in drei Zyklen: Entwicklung einer grundlegenden Struktur, Integration der funktionalen Anforderungen und Integration der Qualitätsattribute. Für jeden Zyklus werden das Ziel, die Wahl der Architekturelemente, der Entwurfsentscheidungen, der Zuständigkeiten, Schnittstellen, Analysen, Diagramme jeweils mit expliziter Begründung gegeben. Die Evaluation erfolgt angelehnt an die *Architecture Tradeoff Analysis Method* nach der *Lightweight Architecture Evaluation*. Diese ist nicht Bestandteil der ursprünglichen ADD Methode, konnte aber als eine Feedbackschleife in die Methode aufgenommen werden.

Zur Umsetzung der Architektur werden die Anforderungen durch eine Umfeld- und Bedarfsanalyse ermittelt. Die Umfeldanalyse basiert auf der Auswertung von nationalen und internationalen Umfragen, Konferenzberichten, Fallstudien und Beschreibungen von *Best Practices* aus öffentlich geförderten Infrastrukturprojekten. Ein Teil der Bedarfsanalyse kann als Grundlage aus dem Umfeld geschlossen werden, zum anderen Teil werden spezifischere Bedarfe aus Einzelprojekten und dem Prototyping gewonnen. Mit Hilfe des Anforderungsmanagements werden diese Erkenntnisse in Qualitätsattribute und funktionale Anforderungen übersetzt. Die Gesamtheit der Ergebnisse aus den Analysen zusammen mit den Erkentnissen des Forschungsstandes und des theoretiscen Hintergrundes bilden das Wissen der Domäne.

Contents

1 Introduction

This chapter motivates the pressingness of dedicating research to the more and more acknowledged discipline of Research Software Engineering and its integration into existing workflows of Research Data Management. The chapter outlines the objectives of the work at hand, specifies three concrete research questions, and presents the structure of the thesis. The research questions fall in three categories: requirements management and field analysis, software quality attributes, as well as architectural design. They are briefly explained and directly derived from the objectives.

1.1 Motivation and Objectives

Research Software Engineering (RSE) is in the process of becoming an recognized research field in its own right demonstrating systematic differences from conventional Software Engineering's (SE) methodological inventory and objectives (Johanson and Hasselbring, 2018). The more well-established field of research data management (RDM) seems to haven taken a similar evolutionary path as it emancipated itself from information and library science. Intuitively, RSE and RDM share a natural connectedness rooted in the research activity, however, consistently divided apart by the justified requirement of a strict separation of data and data creation. Independent from the necessity of keeping program and data apart, the interlacing between the two is particularly apparent for research software and research data (Barker et al., 2021). In extreme cases, the data can only be correctly processed and understood with its originally designed software. Thus, there is good reason to include research software into the best practices of the more advanced RDM infrastructure and figure out which resources could be commonly used and whether it may involve the phase during the software development process as well as the finished software product (RfII, 2025).

Modeling, Simulation, and Data Analytics, *Technology Research Software*, and *Research Infrastructure Software* are the three top-level categories of RSE more recently suggested (Felderer et al., 2025: 21). This thesis falls into the last category because it generalizes away from a concrete case of making a specific research software sustainable, but focuses on the environment and conditions, in which possibly all kinds of research software

solutions can be kept FAIR. The awareness of FAIR research software and its advantages have been growing constantly over the last years, and so is research into this area (Chue Hong et al., 2022). A possible reason for this development could be a pulling effect due to the close interlocking of research software and research data whereas the special status of research data has drawn attention to the creation of the data. In fact, a decisive competitive advantage among universities and other research facilities was attributed to RDM. It seems rather a question of time when research software follows suit. Formally, data and software are kept strictly separate, indeed as prescribed by nearly all design principles. Yet, on closer inspection, it becomes clear that processing data always involves implicit knowledge in the program code, which could as well be added to the data. As practitioners report, this is even more drastically the case for research software: research data is best understood from the software that created it (Johanson and Hasselbring, 2018). So it comes as little surprise that concepts of sustainability, such as FAIR or the early usage of data management plans (DMP), originally developed for RDM are also applied to RSE. In this line of argumentation, it would also be plausible to look at other RDM best practices and find out which of these are also advantageous to RSE. More specifically, one could ask how it is possible and to which extent it is advantageous to incorporate research software into an existing RDM infrastructure.

The aim of the thesis is to give an elaborated answer to this question. It will consider the requirements of research software engineers and research data managers alike, but only include the field analysis, i.e. the institutional conditions of RDM as the incorporating infrastructure. Moreover, additional quality attributes that follow from the requirements are presented and possible contradictions are discussed. The specification will be restricted to architectures at the level of components and use cases modeled in UML. For the more crucial interfaces, concrete proposals will be provided. All of this will contribute to an realistic evaluation of the feasibility of this project, if the respective resources are allocated and the decision to implement it is made.

1.2 Research Questions

The above motivated objective, the integration of research software into RDM services, will now be translated into several more concrete research questions. There are three areas of impact that come to mind rather intuitively when thinking backward from the end to the beginning. The final result should be a realistic architecture interlocking independent research software programs flexibly with available repositories and storage infrastructures. The important point to consider here is the difference between processing data that is subject to change and data that is agreed to stay as is. The architecture is predominantly determined by the quality attributes of both the software produced by the researchers and the software engaged for the RDM infrastructure. The challenge here is to find the set of common qualities and the set of idiosyncratic qualities. The quality attributes follow from the requirement analysis of the users, a field analysis is crucial to involve any other relevant variables from the environment, such as the legal, economic, or ethical restrictions of an academic (public) institution.

Having now arrived at a possible starting line, the three areas – requirements and field analysis, architectural drivers focusing on quality attributes, and the software architecture itself – crystallize as the pivots of the thesis. These can now be concretized and put into specific research questions.

1.2.1 Requirements and Field Analysis

Software requirements management claims to be the point of departure of any actively managed software development project (K1 Pohl, 2010; Sommerville, 1998). Requirements are often collected from use cases documented and discussed in field reports at RSE and RDM conferences. These collections are the primary sources which will be selected for an ideal model of research software and RDM infrastructure. Indeed RDM infrastructures are implemented in research environments such as universities. A field analysis of the institutional setting brings to light further impact factors such as regulations, procedures, and restrictions specific to public organizations. They also will have to be considered and may extend or narrow down the set of requirements due to feasibility. For the field analysis the model case of a German university will be assumed, for which the local arrangement are ignored, but federal regulations of privacy, data protection, and copyright laws are taken into account. Both excluding and including requirements will be reasoned on the basis of the field analysis and consolidated accordingly. To put all of the above in an expressive question, one could say: Which requirements are important to consider for the integration of RSE into RDM services?

1.2.2 Quality Attributes

The essential method, which is used to arrive at an appropriate architectural design, is called *Attribute-Driven Design* (ADD) (Cervantes and Kazman, 2016: pp. 44). Central to this approach are the intended quality attributes that are identified from the field and requirement analysis. Whereas both the design purpose and the primary functionality follows from the topic of this study, the quality attributes are more tricky to find out. Associated with the functional requirements, each quality attribute is paired with a stimulus-response pattern that is testable and falsifiable (Cervantes and Kazman, 2016: 20). Moreover, the attributes can be prioritized by a predefined ranking system in each scenario and depicted in a utility tree. From the field analysis in the first place *constraints* and *architectural concerns* as the remaining architectural drivers will be identified. To keep the focus on the key feature of the method applied here, the research question is simplified to the quality attributes. It reads: Which quality attributes follow from the requirements? It is important to note that all other drivers will still be included in the analysis.

1.2.3 Architecture

From the literature on RSE and RDM published so far, there are no reports on specific blueprints, i.e. reference architectures, one could use to approach the design purpose. In addition, there is no external software available that would meet the objectives of a sustainable integration of research software into the RDM service although the infrastructural set up of RDM services is conceived here as external components as a simplification in the design process. Hence, applying design patterns and tactics is to go about methodologically. The task consists of proposing the best trade-off between requirements of different stakeholders (Bass et al., 2022: 34), but also considering shared resources and particularly interfaces required for integrative software solutions. The iterative process suggested in ADD comprises seven steps that will be passed through several times. These steps will be documented in this thesis. To phrase it as

a research question, one could ask: How can the primary functionality and quality attributes identified for research software integration be converted into an overall architectural style for RDM services?

1.3 Structure

Chapter 2 is dedicated to the theoretical background. It comprises all major concepts prevalent in RDM including the explications to the research data life cycle and its relevance to any integrative solutions. The same will be done for RSE revealing related research whereas research in RSE is clearly differentiated from RSE itself. The very properties of research software are described here as well as its relation to research data. A second part consists of short explications on the applied method. All of this serves as the requisites for the next four chapters (3 4, 5, 6), in which each of the research questions will be answered. Thus, chapter 3 deals with the requirement analysis, in which the elements of a need analysis and field analysis are integrated. Chapters 4 and 5 are dedicated to the design process. These chapters build on each other since each of these is needed to follow the conclusions of the next. Chapter 6 on the evaluation also contains an extended discussion on the proposed design decisions including an appreciation of the architecture's limitations and advantages. The thesis closes with a conclusion in chapter 7 pointing out possible prospects and a final assessment with regard to RSE and RDM.

2
Theoretical Background

This chapter provides the background on RDM and RSE. All necessary definitions, theories, and related work that will be used later on are treated here. To give the right context and domain knowledge without neither requirements nor the architectural decisions can be understood and certainly not developed, important concepts in this domain such as FAIR principles, the research data life cycle, properties of research data, RSE research, its forms as well as the general requirements of public institutions and research software are elaborated. In addition, the methodological procedure of Attribute-Driven-Design 3.0 and the chosen approach on requirements engineering including very short explications on the foundations of field and needs analysis are provided.

2.1 Introduction

This chapter covers two broad theoretical topics: domain knowledge and methodology. To the first topic, the concept of RDM and RSE, the status of data and software within their realm, their institutional peculiarities, general requirements as well as the state of research of RDM and RSE is given. These sections set the stage for making design decisions in chapter 5, but they also give the context needed here. It is probably the simplest way to minimize implicit knowledge and add practical lores. This is particularly necessary to carry out the field analysis in chapter 3. The second topic focusing on the technical methodology will focus on Requirements Engineering (RE) or Requirements Management (RM) and Attribute-Driven Design (ADD). The former is the basis for chapter 3 and chapters 4 through 6 heavily build on extended knowledge of ADD. Thus, all ground work required to follow the rest of this study is laid down here.

The available literature will be shortly discussed at the beginning of each section. Yet, the situation of the established sources could not be farther apart. RM and ADD as acknowledged fields in software development harbor a wealth of standard textbooks, high ranked peer reviewed journals, established specialized literature, and indeed recognized and well-attended conferences. For RSE the situation is different. As a discipline still struggling with emancipating from a reputation of producing *patchwork*,

the community can refer to little more than a yearly conference with proceedings, first descriptions of RSE best practices published online and memoranda of understanding. One can observe though that a lot has started to move, structures are set up and communities are built with the help of publicly funded infrastructure projects such as *NFDI*. In terms of literature, RDM could be viewed somewhere between RM/ADD and RSE. Published books take the form of anthologies, collections, and manuals that can be better characterized by heterogeneity than coherence. Within the context of library studies, where RDM is claimed to be a sub-discipline, the situation looks more promising. It could than be paralleled to RSE being a special case of Software Development (SD).

2.2 Research Data Management

This section recapitulates the essential concepts below from the few resources specifically published to this topic (Büttner et al., 2011; Kruse and Thestrup, 2018; Clare, 2019; Putnings et al., 2021; Mons, 2018). When going through the literature, it becomes apparent that there is no standard publication available, which more or less all authors would refer to. Neither extended discussions on RDM issues can be observed in the established national or international journals ("Harvard Data Science Review," 2025; "Data Science Journal," 2025; "Bausteine Forschungsdatenmanagement," 2025). Sources take the form of anthologies and case studies of selected RDM topics. A coherent presentation or textbook as a unifying element is missing. The textbook of the German RDM community, for example, comes in the form of a live-writing project (Helbig and Neumann, 2016). The online resources for the German community cover most of concepts as well. They come as an interactive website ("forschungsdaten.info," 2025) or a simple wiki ("Wiki Forschungsdaten," 2025).

In this thesis the definition from BMFTR (2025) for RDM is taken as a basis.

> Das Forschungsdatenmanagement umfasst alle Maßnahmen zur Organisation, Bewahrung und Dokumentation der erhobenen Daten. Es erstreckt sich über alle Stadien des Forschungszyklus, von der Planung eines Forschungsprojekts über dessen Durchführung bis hin zur Veröffentlichung der Ergebnisse und der Bereitstellung der Daten in Repositorien oder Datenzentren, um die Nachnutzung durch andere Forschende zu gewährleisten. Ein nachhaltiger, offener Umgang mit Forschungsdaten fördert die Transparenz, Reproduzierbarkeit und Nachnutzung der Ergebnisse. Das Forschungsdatenmanagement ist wichtiger Teil der guten wissenschaftlichen Praxis. (BMFTR, 2025)

It reads essentially the same as "forschungsdaten.info" (2025) and so it is plausible to assume that there are no substantial deviations what RDM is and comprises. Yet, the latter mentions more explicitly that "'structured measures can be taken at all points of the data lifecycle to preserve the scientific validity of research data, to maintain its accessibility by third parties for evaluation and analysis ..."' [1] Although not explicitly mentioned, such *structured measures* would involve the incorporation and maintenance of research software.

1. https://forschungsdaten.info/praxis-kompakt/glossar/#c269821

2.2.1 FAIR Principles

FAIR is the acronym for findability, accessibility, interoperability, and reusability. These concepts are central in data management and essentially all RDM activities aim to achieve FAIR. They cut to the heart of the long term RDM objectives. If heeded appropriately, sustainability follows as a consequence and it promotes the concept of *Open Data*. A very similar initiative has taken place for research software engineering (*FAIR4rs*). To understand how they interrelate, why they are relevant and that other concepts such as the data life cycle built on top of them, the FAIR notions will only be briefly introduced here to avoid confusion and repetition in the section to come.

Findability

Findability suggests that a collection of data is sufficiently described. From the descriptions, all properties of the data must be immediately apparent: type, size, representativeness, purpose, provenance, and many more. The descriptions about the data is usually referred to as meta data, which is generally available independent from the actual access restrictions of the content data. Findability also depends on how well known the data is. The reputation and trustworthiness of data repositories and the inclusion in world-wide retrievable catalogs promote the findability of data sets. Thus, the concept of findability prescribes meta data standards and unrestricted admission in all relevant indexes.

Accessibility

Knowledge of a specific data set is the first prerequisite of being able to use it. Nevertheless, the data may be restricted in access for some groups but not others, embargoed til a predefined time span has passed, or simply never accessible. While the idea of open data is most desirable, but in its entirety an ideal, there may be good reasons for exceptional (and very few) cases to keep research data closed. As a general guideline, research data produced in public institutions should always be open in the long run. For the time of qualifying works if the competitive situation between individuals or teams would lead to unfair exploitation, the data can be restricted for the time the articles are written, but not published. There are also some very costly and research intensive co-operations with industry leaders that amortize their investment of innovative products by keeping research results closed as it is often the case for pharmaceuticals. These are cases of business profitability. Research data, which conflict with the legitimate public interest or endangering public security such as data on effects and composition of synthetic intoxicants or weapon technology have good reason to be restricted to a smaller user community. Data that cannot be anonymized although it is required, will also fall in this category. These exceptions, however, should not generalize to the vast majority of research data, which is kept from public access due to greed, fear, or shame. Parallel to open software, the quality of open data improves rapidly since other researchers extend the data, find errors, correct, and standardize it. At the bottom line, for the research community, the advantages of full access outweigh by far the disadvantages.

Interoperability

Interoperability aims at data use independent of technical restrictions such as specific requirements of the hardware and software: platform, operating system, data format,

version of the executing program. Indeed all the impacting variables may interact with each other and it is hard to foresee future developments and dependencies of a mighty software industry. Still, one can observe a steep tendency towards non-proprietary formats, software, and standards in the open-source community. The advantages of platform independent operation of data and software also could no longer be ignored for big businesses. Web-based protocols and applications as a driver of standardization contributed tremendously to a harmonizing effect and overcoming some of the power relations.

Reusability

This concept describes the recyclability of digital data without attrition. Indeed, data may become obsolete but not as a function of usage frequency, which rather leads to an increase of use, as for actuality and outdatedness due to technological advancements. Two main issues determine practical data reusability in the first place: knowing that suitable data exists (findability) and being authorized for usage (accessibility). If good documentation on all specifics of the data is also available, reusability is met.

2.2.2 Research Data Life Cycle

The research data life cycle (RDLC) is a graphical visualization of the main fields of RDM work (figure Figure 2.1). There are many variations of the same idea. Some put more emphasis on the data aspects; others explicitly mention managerial aspects such as planning. Essentially they all involve data collecting, analyzing, sharing, and reusing. The cycle reveals a sequence, in which data services are usually applied to a research project. In addition, it implies repetition of the services, i.e. a cyclic process, although it must not necessarily be the case. Some of the services include further repetitive processes that are not visualized[2].

Planning

Planning in RDM is more than planning the data collection and analysis processes as usually done in research projects alone. The focus in RDM is on sustainability and this means that long-term storage, archiving technologies, and financing their costs play an important part in the planning process. The extent of what sustainability means here becomes apparent when it is put into practice; yet it still has to receive full consideration in planning. Within an approach of sustainable data, data scientists have to guarantee that the data is fully usable in the future. Problems that often occur with proprietary data formats is, for example, that they are no longer readable after only a few version changes of the software needed to view the data. To circumvent these pitfalls, data curation must be carried out and a respective documentation must be produced. Realistic costs for the transformation should be evaluated beforehand. And this often leads to considering alternative ways of data production or, if possible, reusing existing data mostly with consequences for the research questions. So in the planning process the entire research design is made explicit with regard to data processing. Within the planning process, the researcher goes through all stages of the research data life cycle in figure Figure 2.1 and anticipates possible future challenges like ethics, data sharing, or legal requirements.

2. https://www.dcc.ac.uk/guidance/curation-lifecycle-model

Figure 2.1: The Research Data Life Cycle captures all relevant stages that scientific projects undergo.

Elaborated tooling may assist the researcher in this process, which produces a plan after going through a questionnaire. Such a plan is continuously adapted.

Collecting

At this stage the technical knowledge prevalent in the specific discipline is crucial. The researcher will make decisions regarding the method: observations, experiments, simulations, text analysis, and many more. It is also the stage, in which FDR is gloomily underestimated for two reasons: ignoring available data sources and data authorization as a legal requirement. Missing the reuse of existing data is generally due to lack of transparency and clarity as well as embargoed data. As RDM further unfolds its measures towards data sharing, open data, data accessibility, and standardization, this problem will more and more diminish. At the same time, legal standards for receiving permissions to process and use the data differentiate more and grow in size. Together with a growing awareness that behavioral data be a strong currency in the future, the legal prerequisites grow in size.

Processing

Having finished data collection, further processing is necessary to bring the raw data in the form needed for analysis. For quantitative studies statistical methods are usually applied to make the data accessible to the human brain. Before that, pre-processing may be indispensable to standardize and fulfill the requirements of the statistical method. Outliers, obvious miscalculations, incorrect classifications or misassignments have to be

treated and documented. These kinds of cleaning is also incurred for qualitative data. Transcriptions may be produced or the raw data must be digitized or even translated. In general, the data has to be validated in terms of integrity, completeness, and consistency. Anonymization falls within the scope of the processing stage. A plethora of commercial and open source tools are available to assist the researcher in data processing and it is probably impossible to give exact numbers on the researcher's self-programmed software solutions.

Analyzing

At this stage, the actual research results are generated. The researcher aims to discover patterns in the data; that is, combining, reconciling, or delimiting the data with previous knowledge and theories. A thorough description precedes the interpretation of data before the result might unleash an explanatory potential. Especially at this phase, the question of data retention for the time of the project must be clear. Storing data that still undergo substantial change have other requirements with regard to access time and frequency, embargo, or identification. Beyond data repositories, there are no general software solutions that are able to deal with the opposing prerequisites of active data management and archiving altogether. Thus, one can observe a few compromising solutions, in which active data retention is administered with foresight to its later storage. The earlier the standards of the long-term storage are considered in active data management, the easier and cheaper the subsequent transfer to the storage system will become.

Publishing

Publishing research data is predominantly digital. Texts that only exist in print have become rare to non-existent. But even digital text publications state less of an infrastructural problem to RDM than large data collections that need both up- and cross-scaling. Data publications can be part of a journal article, in which e.g. the experimental data is kept together with the text, or stored in discipline-specific data repositories, in which meta data standards of the respective research community are available. General purpose repositories have the disadvantage that compromises of the description of the data have to be made, i.e. the meta data may not fully meet the needs of the data to be published. Exact copyright definitions such as Creative Common licenses and access control schemes opting data embargoes with expiry dates or well-defined data sharing conditions are essential services of all data repositories. Cataloging and dissemination are no less important and a crucial criterion for the quality of the repository operator.

Archiving

Depending how the publishing process is fleshed out, archiving turns out to be a heavy burden or nearly effortless. By definition, archiving procedures involve data migration to suitable formats and transferring to appropriate media, creating backups and saving data, as well as defining all meta data and preparing documentation. In modern storage infrastructures, the archiving process can be handled in the background while the frontend functions as a publication interface. The advantage of such an archiving solution is that the above mentioned challenges of formatting, transferring, saving, and documenting is already done in the publication process and has not to be redone.

Being aware of which formats are susceptible to change, have proprietary restrictions or low acceptance, is part of a process of balancing the cost of data procurement and data curation. It depends on each individual data project. As a general rule of thumb, platform-independent and open formats should be preferred. It cannot be stressed enough that detailed data documentation carries the highest savings potential for data reuse in the long run. The same is true for extensive meta data definitions and the use of the standards accepted within the research community for findability. Also, for interoperability, the choice of format is crucial.

Reusing

Re-using research data is thought to carry the highest savings potential. There are hundreds of cases reported in the literature, in which very expensive surveys collecting the same data were carried out multiple times, simply because the data were not accessible or not known. So in reusability all other FAIR principles come together to its ultimate goal. To approach this goal, domain-specific data repositories are the best possible solution as of now because findability and accessibility are inbuilt. For interoperability and reusability more responsibility is directed to the contributing researcher, but even here the more fundamental requirements as documentation and open formats can be fostered. Before reusing the data, the user should carefully check the applied methods and procedures from the documentation as well as the completeness. Such a review is supposed to be done with a view to the ongoing research, new assumptions and the possibility of making adjustments to the data. Oftentimes it is necessary to carry out additional calculations or more recent methods. In other cases it will be necessary to extend the data before the new research questions can be investigated.

Discovering

This feature of the research data life cycle is not always included in all versions since it refers implicitly to the stage of publishing and there, more particularly, to the cataloging and disseminating function. As already mentioned there, the quality of the publishing institution, that is, the operator of the repository plays an important role here. Established repositories well known to the entire research community facilitate dissemination as well as the presence in all recognized catalogs.

2.2.3 Properties of Research Data

The *Federal Ministry of Education and Research* defines research data as

> [...] Daten, die im Laufe des Forschungsprozesses entstehen, zum Beispiel im Rahmen von Experimenten, Messungen, Quellenforschungen, Erhebungen, Befragungen oder Daten, die mit einer wissenschaftlichen Methode verarbeitet werden. Forschungsdaten stellen die Basis für Forschungsergebnisse dar. (BMFTR, 2025)

Again, this definition corresponds to the definition used at "forschungsdaten.info" (2025) and other anthologies.[3] As concrete examples are given measurement results

3. https://forschungsdaten.info/praxis-kompakt/glossar/#c269821

and laboratory values, texts, results from surveys, field notes, audiovisual documents, objects or even the software used to process the data. From the given examples, some properties of research data can be derived; other properties originate from practical handling and legal provisions.

The definitions above imply that all scientific disciplines and their methodological set up are included. From this, it follows as a first important property *heterogeneity*. It implies that the methods to make the data accessible must be diverse as well. The option of an all embracing software that displays 3D-Models of ancient pottery, complex annotated text of Old Mongolian, 16th century Bantu rhythms, or sequences of DNA base pairs is not to be seen despite all progress in this area. Thus, maintenance of the software enabling accessibility will stay. The heterogeneity of relatively small amounts of data (long-tail problem) and in part a divergence from agreed standards comes down to managing a efficiency problem such as huge amounts of data have to be handled within an acceptable time frame or many different applications with diverse functionalities have to be handled with an acceptable number of resources. Key to the management here is to use scaling effects.

As a second property, research data is the result of a scientific process which strives to come as close to an objective truth as possible. Because it is corrected and refined as an ongoing process of the research activity, the probability of being true is higher as data that is produced otherwise. Without denying exceptions, the quality of the data in duration and average is also higher.

Third, from research data it is impossible to say that it is no longer needed. Again it is a property that is not valid in absolute terms but as a probability. Actuality and frequency of use may be indicators of future necessity, yet there is no guarantee that sudden technological leaps turn useless data of the past into an asset. Some data depending on time or the conditions of a certain time (climate) cannot be recreated. At its extreme, even a falsifiability check is only valid in the given context and for agreed assumptions. Both, assumptions and context, may change over time.

Considering the legal properties, it is important to note that research data by the above definition is not congruent with the legal interpretation of the German copyright law whose essential principle is the freedom of facts and information that applies to data in general. To avoid conflicts with ownership rights, some typical research data outcomes such as texts, software, pictures, or audiovisual documents are not included in the definition of data by the legal reading of the term. Essentially, the definition of data that the German legislation applies differs from the definition applied in RDM. Whereas any kind of data that is produced in the research process with or without other information sources is defined as research data in RDM, the copyright law treats data sets differently that is processed in a certain way (e.g. tagged corpora) or enriched with additional material (Wünsche et al., 2022).

This leads to a fourth property. Raw research data is not possessed by anyone reasoned by the fact that data in the sense of the German copyright law does not underlie the right of ownership. For reasons of ignorance, property rights are often claimed for independently collected data. As a matter of fact, there is no legal basis for this (Kreutzer and Lahmann, 2019: pp. 42). Part of the confusion may be that according to § 87a UrhG databases themselves are protected by the copyright because they represent a significant investment. The amateur is inclined to think that data collection is a significant investment, too, at least as much as the retrieval of data in databases. In legislative terms, a difference is made and the argument of cost expenditure is not

referred to the collection of data. However, data enriched with additional material such as pictures, descriptions, or other textual information may fall under the copyright law.

Last, research data share many characteristics of public goods. In economic terms, the procurement and provision of research data pay off for the entire society rather than for the individual user. In other words, the investment in research data is profitable for the sum of collected data in the long run. Analog to the investment in science in general, private investments are rare for the low attributable benefit to a certain data.

2.2.4 Institutional Requirements

From the above properties of research data follow some important requirements for public institutions that differ substantially from private organizations or personal use and hence they prescribe additional requirements having to be taken into account for designing software. From the similarity with public goods and the generally missing ownership follows the principle of openness. Apart from the exceptions listed in section 2.2.1, research data have to be usable for everyone with lowest acceptable barriers. This has important consequences for the usability attributes in software design. Besides this, other responsibilities regarding longevity follow from the third property, missing copyright, mentioned above. Together with the heterogeneity property, it poses high standards for the quality attributes of maintainability. Since research data strive towards a general truth, its accessibility must be independent from political influence and the political situation.

2.3 Research Software Engineering

2.3.1 Properties and Forms of Research Software

> Research software is software that is designed and developed to support research activities. [... It] is typically developed to meet specific research needs, and often has unique requirements that are different from standard commercial software (Johanson and Hasselbring, 2018; Hasselbring et al., 2025: 59).

The classic among RSE cases describes a researcher developing program with the only goal of getting a very specific set of data. It is this set of data, which brings reputation and opens up further career opportunities. The software is worth little more than keeping it as a must for documentary purposes. In many documented cases, it is not maintained and in few cases, it is deleted altogether. Although the advent of RSE shifts the focus of mere software functionality to other areas in the creation process, such as readability and comprehensibility of code, version control, licenses, documentation and many more, the properties of research software differ substantially from software serving the needs outside the research domain that have to fulfills many other quality attributes if the product should survive in the market. In principle, the ideosyncratic nature of the data to be generated or collected by research software forbids a commercial use for the low number of potential customers. How exactly RSE differs from general software engineering is investigated in the emerging RSE research.

As put forward above, the definition of research software leaves room for interpretation. From observation, it is plausible to make a broad categorization of research

software, that is, outside (commercial) and in-house development. While this observation is easy to prove right, it is not detailed enough to get to know the domain specifics. Hasselbring et al. (2025) suggest a multidimensional categorization of roles, developer, readiness, and dissemination. The purpose of such categories are plentiful. Yet, with sight to acquiring domain knowledge such categorical aggregations of research software reveal quality requirements, institutional guidelines, stakeholders, necessary metadata, or software artifacts. According to this approach, the three main categories are role-based (1. *modeling, simulation, data analytics*, 2. *technology research software*, and 3. *research infrastructure software*, (Felderer et al., 2025)) and can be subdivided into further subcategories. For the first category, modeling and simulation, data analytics, software analytics, integrative analytics, and scientific visualization are specified. The second category contains four refinements: hardware related, software related, human related, and process related. These may refer to another sub role that belong to the level of readiness-based categorization, called Technology Readiness Level (TLR). For the research infrastructure software, to which the proposal of this thesis belongs to, eight more categories are enlisted: control and monitoring, data collection and generation, pipelines and tools, libraries, laboratory notebooks, data management, software management, as well as collaboration and publication (Johanson and Hasselbring, 2018: 61). These subcategories resembles the main fields of RDM, which are displayed in the research data life cycle (figure 2.1). Hence, the close interconnection with RDM is clearly visible here again. Further categories dissemination-based and developer-based. Dissemination-based categorization refers to open source, closed source, or software as a service. Finally, the developer-based category comprises individual researcher, local research group, project group, community, and contractor.

2.3.2 RSE Research

Felderer et al. (2025: 22) suggest several research questions specific to RSE research.

- What is specific about RSE compared to other SE specializations?
- How to organize software-centric scientific processes?
- How to integrate SE techniques such as requirements engineering, architectural modeling and automated testing in the research software development process?
- What additional tools do RSE practitioners need?
- What is the role of generative AI in RSE practice?

Put briefly, research is addressed to the specialization, processes, integration of requirements engineering and architectural modeling, tools, skills, and indeed the role of AI in RSE. Requirements management is of particular interest in RSE research since although it is generally acknowledged that requirements need to be known upfront, it is also clear that the nature of research projects is that functional requirements are unknown at the beginning of a project, emerge slowly during the project, keep on changing until towards the end of the project requirements can be specified to a sufficient degree of detail.

Besides the above research questions, Druskat et al. (2024) point out that further research is carried out in areas of RSE community building comprising areas such as developing a common language, elaborating security and usability issues specific

to research software as well as life cycles and categories of research software types. From the given explications one can see that this branch of research is still in a phase of orientation and discovery.

While sustainability is a major objective in RDM, it is not listed among the main fields of RSE research in the listing of Felderer et al. (2025). The categorization, so it is claimed, "may help to emphasize which software is critical, highlighting the importance of its maintenance and continued development for its continued functionality. By highlighting this importance, we seek to contribute to an enhanced awareness of the ongoing support and resources required to ensure the longevity and sustainability of research software" (Johanson and Hasselbring, 2018: 66). From the perspective of RSE-RDM-integration, this could be, however, the most important question, namely how to keep research software (even in isolated environments) running for as long as possible. A decade ago, the report on *research software sustainability* reveals that across Europe (Germany, Netherlands, United Kingdom) and outside (Australia, Canada, United States) the importance of solutions and building infrastructures is widely agreed upon and recognized while long-term maintenance of research software is not supported by public funding schemes (Hettrick, 2016: 22). More recently, some progress can be observed, yet the implementation of solutions looks very ideosyncratic (Katz et al., 2021). Indeed, the discussion on sustainability illustrates how complex the challenge is. Thus, the topic of keeping research software sustainable is unlikely to be solved in this thesis either, and consequently no solution should be expected in the architecture. Solving this fundamental problem, if it can be solved at all, remains a question for future research and large data infrastructure projects. For the project at hand the common ground is the data level. It is feasible and it is where the objectives of research software according to its own definition and research data management come together.

As already pointed out in the introduction (1.1), the work at hand falls in the main category of *infrastructure software* (Felderer et al., 2025; Hasselbring et al., 2025). Related research in this area orbits around the idea of quality control. Quality control comes in different flavors: for software architecture and how to measure it (Druskat et al., 2025), for the data curation workflows and its data in various domains (Hedeland and Ferger, 2020; AtMoDat, 2025; QUEST, 2025; GREI, 2025). In addition, software implementations for data curation workflows to be integrated in established repositories ("Conquaire," 2019; Invenio, 2025b, 2025a) and specific funding programs of public institutions (BMFTR, 2018) are available.

2.3.3 General Requirements of Research Software

Researchers usually do not have any professional training in software engineering (Felderer et al., 2025). They perceive software as a tool to get the data proofing a thesis. Consequently, functional requirements are defined as the main objective of the software. Even these are likely to change within the research process, non-functional requirements such as performance, reliability, or readability are at best implicitly implemented as side effects, i.e. if the program turns out to be slow some other data structure is tried out as an alternative.

From the point of view of integrating research software into a RDM infrastructure, it is directly comprehensible that the functional requirements of single applications play hardly any role in the architectural strategy. The functional requirements are best considered as encapsulated units that "do something". Rather it is the non-functional

requirements, i.e. the quality, deserving attention in the integration process (Chung, 1998). For quality requirements the above described FAIR principles state something as a minimal agreement that all data management must fulfill. In how far these findings can be applied to research software is still subject of the ongoing debate, but first results give good prospects that some adjustments are necessary. *FAIR4RS* (Chue Hong et al., 2022) spell out what the FAIR data principles would mean for research software. This study is based on a collection of case studies and other contributions (WG, 2021) as well as the expertise of 500 members from 110 organization in 34 countries (Lamprecht et al., 2021). As a brief summary, the concepts of findability (F) and accessibility (A) show high degrees of similarity and can be adopted in research software. Interoperability (I) and reusability (R) would need adjustment.

> We define interoperability as how information (data, metadata, application programming interfaces (APIs)) is exchanged. [...] we define reusability as both usability (the software can be executed) and reusability (it can be understood, modified, built upon, or incorporated into other software) (Katz et al., 2022)

Thus, the redefinition of *I* and *R* adds another quality attribute – usability – to the existing interoperability and reusability. What findability and accessibility specifically mean for RDM integration will have to be spelled out and possibly other attributes will have to be derived there as well.

Besides narrowing down the requirements to non-functional requirements and admitting *FAIR4RS* as the basic set of requirements, a third question is to which extent the quality requirements of research software should be reflected in the architecture of the incorporating RDM infrastructure. This question has not been addressed so far in the pertinent literature. Yet, it will be possible to give good reasons to include additional attributes and make a clear distinction between the requirements of the research software and the requirements of the infrastructure. As referred in the subsection on institutional requirements (2.2.4) above, best practices from RDM institutions across all countries reveal the importance to include quality attributes such as maintainability and integrability as well as derivations from the FAIR and FAIR4SE principles, which are harder to manage, but crucial to the acceptance of an research application. Other quality attributes such as performance should be left to the single applications and the hardware used there.

2.4 Methodological Procedure

This section sketches briefly the applied method to arrive at an architectural design. As depicted in figure 2.2, the first step will be to determine the requirements from all stakeholders, which includes a need analysis as well as a field analysis (chapter 3). From these results and the context elaborated above, the architectural drivers will be derived (chapter 4) in a second step, that is called here architectural design, whereas an iterative process is run through to work off all architectural drivers. The third step involves the production of an elaborated documentation, which is done in parallel with the thesis at hand. As a last step, the architecture will be evaluated. The implementation is restricted to partial prototypes, but in its entirety is left to follow-up projects.

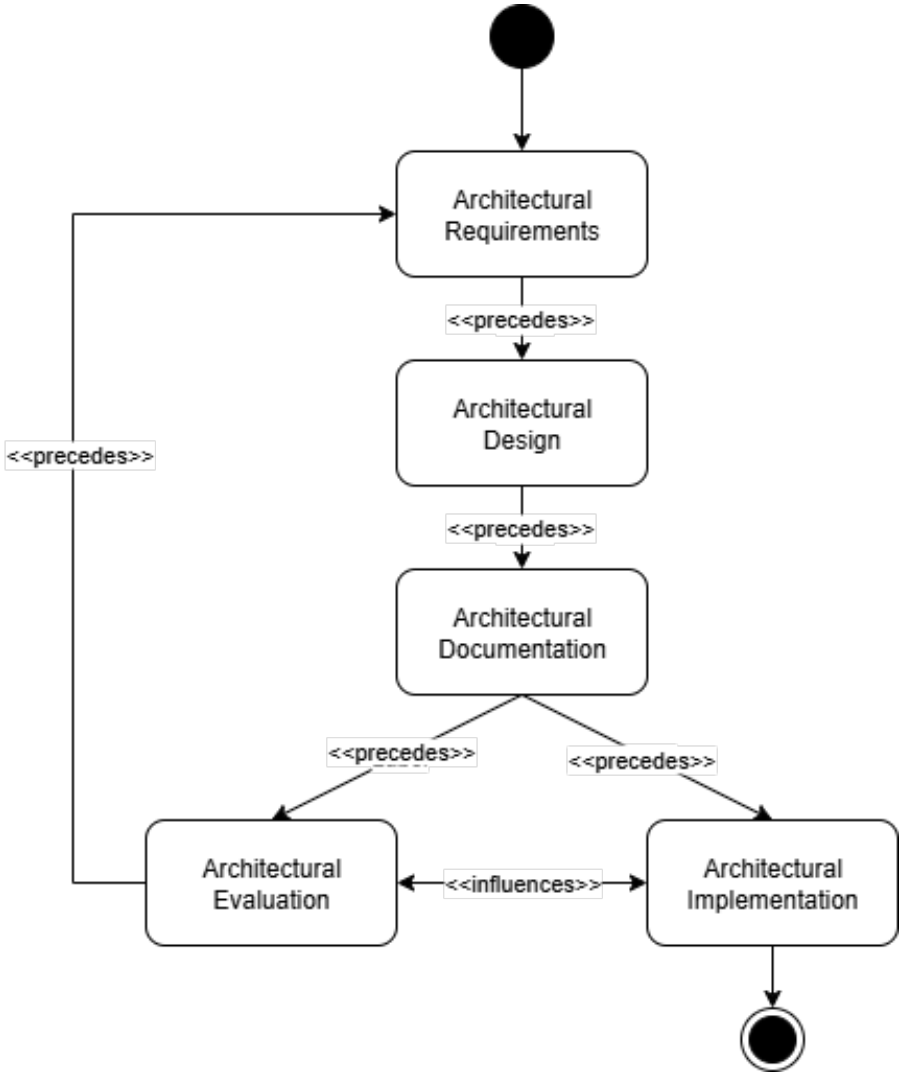


Figure 2.2: Software architecture life-cycle activities (Cervantes and Kazman, 2016: 5)

2.4.1 Requirements Engineering

Requirements engineering follows in general after the fields and needs analysis since this discloses stakeholders and their needs. The initial phase is characterized by concretizing the needs of stakeholders and transferring those into formal requirements. These requirements are functional – what the system should do – and non-functional – how should the system be (Whalen et al., 2013).

Maalej and Thurimella (2013) recapitulate different approaches and various definitions of requirements engineering (Davis, 2003; Sommerville and Sawyer, 1997; Nuseibeh and Easterbrook, 2000; Aurum and Wohlin, 2003; K Pohl, 1996; Hofmann and Lehner, 2001; Yang et al., 2008). A common and often engaged definition in the context of systems engineering reads that

> Requirements engineering (RE) is the branch of systems engineering concerned with the *desired properties and constraints* of software-intensive systems, the goals to be achieved in the software's environment, and assumptions about the environment (Maalej and Thurimella, 2013: 2).

Conceptualization such as these in systems engineering all culminate in a knowledge view of requirements and could therefore be thought of from this perspective. Emanating from a concept of knowledge management and perceiving requirements knowledge as consisting of *the implicit or explicit information that is created or needed while engineering, managing, implementing, or using requirements* (Maalej and Thurimella, 2013: 7), they define requirements engineering as managing requirements knowledge.

> Managing requirements knowledge is about efficiently identifying, accessing, externalizing, and sharing all types of requirements knowledge by and to all stakeholders, including analysts, developers, and users. (Maalej and Thurimella, 2013: 8)

Within an requirements knowledge approach, activities, artifacts and documents, stakeholders, and collaboration are specially emphasized. Activities include the classic tasks requirements elicitation, analysis, specification, verification, and management. Artifacts are often captured in unstructured or semi-structured documents in natural language. This is particularly the case for non-functional requirements (NFR) whereas the functional requirements are documented in use case diagrams or descriptions. Besides texts, mathematical models are used for formal requirements description and visual models such as UML are employed for semi-formal documentation. Also multimedia material enjoys growing popularity (Maalej and Thurimella, 2013: 4). The same is true for distributed development, which puts higher emphasis on collaboration. At the end, requirements engineering is still about figuring out who the stakeholders are and how to know what they exactly think and need. This is where the knowledge part is most apparent.

Requirements engineering as managing requirements knowledge is about acquiring knowledge about the application domain, capturing and using process and product knowledge, acquiring knowledge about new technologies and knowing who knows what as a prerequisite of collaboration. Put briefly, domain knowledge, engineering knowledge, management knowledge, as well as collaboration and how-to knowledge are the central areas in requirements engineering and thus it is about managing requirements knowledge. This way of thinking about requirements engineering will be the basis for the thesis at hand without denying the effectiveness of other approaches (Hofmeister et al., 2005; Hood et al., 2008; Chemuturi, 2012).

2.4.2 Field and Needs Analysis

The setting of an RDM infrastructure is derived from the best practices and a field analysis of different organizations. A field analysis is to get deep and elaborated knowledge of the wider context and domain, in which a software is to be implemented. The field analysis reveals existing technologies and other software for similar problems that possibly could be reused and extended. Best practices, workflows, and existing procedures could be copied or adjusted. Above all, the field analysis unfolds all stakeholders. It is plausible to distinguish different degrees of influence, interest, and knowledge of the stakeholders. The analyst may think of them as several circles around or layers above the core software product. Daily users of a software system have a different perception of usability and functional scope, decision makers may be focused on the cost and inclined to make compromises in usability. A field analysis will not remedy contradicting interests, but it collects, describes, and makes the architect aware of them.

A needs analysis is a notion related to usability requirements as engaged in Human Computer Interaction (Lindgaard et al., 2006). In the last two decades, this idea merged into Human-Centered Design approaches and was broadened to other quality attributes than mere *usability*. Most of the original methods are further applied nowadays. Among them are questionnaires, workshops, round-table talks, as well as analysis and studies of secondary sources. Business goals are derived from the insights gain by the need analysis.

2.4.3 Attribute-Driven Design

Justification for Selection

The relevant literature describes several methods of architectural design. Hofmeister et al. (2007) compare five approaches (ADD 2.0, Siemens 4 views, RUP's 4+1 Views, BAPO, and ASC) and derive a general pattern for designing software architectures (Figure 2.3). Unfortunately some of the aforementioned methods are subject to copyright and so the details were not accessible. There are other design methods such as the Architecture-Centric Design Method (ACDM) or Rational Unified Process (RUP) (Kruchten, 2003), which are not very explicit with regard to concrete steps of action. A method employed at IBM (Eeles and Cripps, 2009) also has little focus on what exactly to do on the iterations and which scenarios can be chosen in which context. Microsoft (Microsoft, 2009) is similar to ADD, but less detailed, which provides little guidance for design decisions.

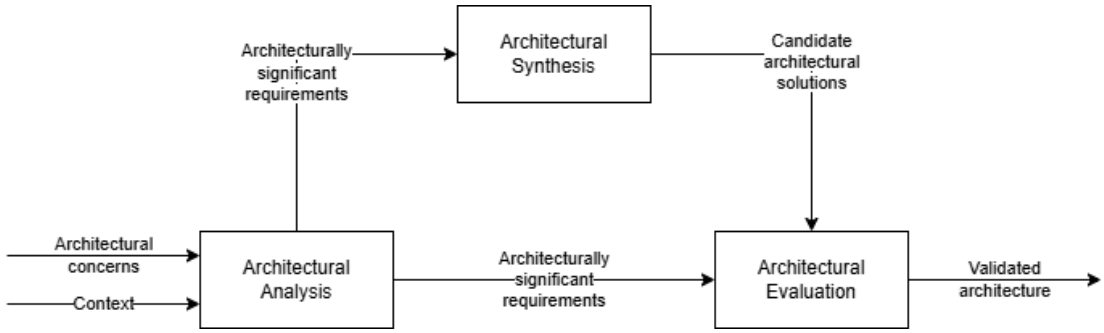


Figure 2.3: Architectural Design Activities (Bass et al., 2022: 162)

To summarize briefly, the decision on which method to employ depends on the level of experience of the architect. For aspiring newcomers, more detail on processes, step-by-step guidance, and less abstraction for design decisions is useful. Additionally, the method must be publicly available without restrictions on utilization. Hence, ADD 3.0 seemed the most convincing choice for the thesis at hand. ADD is also based on the identified design activities in Figure 2.3 that are regarded as success factors in other software systems.

Architectural Drivers

The theory on architectural drivers is based on the explications in Cervantes and Kazman (2016) and will merely be summarized here. The specification for the planned design can be found in chapter 4. There are five drivers that should be considered when constructing any software architecture: the purpose of the architecture, its primary functionality and quality attributes, architectural constraints as well as concerns.

Every software architecture is designed with a concrete purpose in mind. The design purpose could be carried out for a feasibility study that serves to better understand stakeholders and requirements or to simply be able to specifically consult a business. The purpose could also be done for an exploratory prototype that seeks to explore the domain, the technology, or the relevance of quality attributes such as scalability or performance. Another possibility is that the architecture is part of an development process, in which refactoring, work assignments, and a final release will take place. Developing greenfield or brownfield systems impact heavily on the design purpose. Domains that are well-known already carry a lower risk of failure and of encountering unknown events. More architectural work reduces this uncertainty, but also more scrutiny on requirements have to be invested. And lastly, the business goals of the organization are reflected in the design purpose. The organization may decide to focus on reuse and future extensions to bind customers in the long run. Such a business goal may presuppose scalability as an additional quality. Strategic relationships to other products or even personal dislikes might play a role. They all have to be accepted as such because they affect the design process.

"Primary functionality is usually defined as functionality that is critical to achieve the business goals that motivate the development of the system" (Cervantes and Kazman, 2016: 25) and for about 10 percent of the use cases this is the case. The most important reason why to consider primary functionality in the architecture are quality attributes either because of modifiability and reusability expressed in how the functional elements are allocated or because the quality attribute scenarios directly follow from the primary functionality as it is often the case for performance if response times is crucial. It may even exclude functionality. And, false compromises in allocating the functionality accumulates to *technical debt* that sooner or later faces the decision to refactor. In turn, it influences the overall progress and velocity of a software project.

The most significant driver are the quality attributes of a system. Most systems are redesigned not because of failure in the primary functionality, but because they did not meet a quality attribute.

> A quality attribute (QA) is a *measurable* or *testable* property of a system that is used to indicate how well the system satisfies the needs of its stakeholders beyond the basic function of the system (Cervantes and Kazman, 2016: 39). (highlights not in original)

Moreover, quality attributes fall into two categories. Either they describe properties at runtime auch as availability, performance, and usability or development such as modifiability, testability, and deployability. From the debate of the last decades, it became clear that a quality attribute has to quantify as exact as possible (Cervantes and Kazman, 2016: 41) so that prioritization and elicitation can be done. For reasons of clarity, quality attribute requirements are best described as scenarios including a stimulus source, the stimulus, the environment with the artifact, the response and the response measure. There are two techniques, the quality attribute workshop (QAW) and the utility tree, to find out which QAs exist and which are more relevant than others. They complement each other. QAW addresses external stakeholders and utility trees internal stakeholders.

Constraints come in two flavors, technical and non-technical, and in most cases can be hardly influenced by the software architect. Technical constraints include the use of open source technologies or standards needed to interoperate with and integrated into. Some systems have to be backward compatible or handle older versions, which prevent

the operation of a better suitable technology. Non-technical constraints may involve legal requirements, such as the AI act or non-negotiable deadline. Also the qualification of the personal, their ability and availability impacts strongly.

Architectural concerns may be conflated with constraints although their nature is different, i.e. the architects can do something about it. Unfortunately, some of them are tacit and need experienced architects. Four different types are identified. First, *general concerns* comprise problems that naturally happen during the design process: when functionality is allocated to modules or teams, when the code base is organized, or the deployment and update strategies fail. Second, *specific concerns* are related to dependency management, configuration, logging, authentication, authorization, or even adapting a reference architecture to the local application. Thus, they are system-internal. For some instances, these problems result from earlier design decisions. Third, *internal requirements* are requirements of a system that are not specified in any documents because they are possibly taken for granted and come to the fore when unexpected changes occur. These peculiarities may comprise development, deployment, operation, and maintenance procedures. Finally, *Issues* refer to the unforeseen that come to light in the review process. This is typically the case for unpredictable risks that require a change in the design decision.

ADD Design Process

The ADD design process is depicted in figure 2.4. The above described drivers are considered as an input and reviewed in the first step. From there on six more steps follow iteratively for all remaining drivers. For brownfield developments, the outer loop starting with the review of inputs that includes an existing architectural design is also repeated several times.

Review inputs means to make sure that the architectural drivers are correct. There may be the necessity to design prototypes. Usually the results of a QAW and a utility tree are available from which prioritized quality attribute scenarios and primary functionality can be taken. The purpose of the design is made explicit and a design backlog is created, in which each future step will be documented.

With the second step, each new iteration will start with a check-up on the design goal and the decision if the goal is reached. The goal to be reached depends on the driver or the subset of drivers that is selected to be solved. Such a goal could be a attribute scenario or achieving a use case. For greenfield developments in a mature domain, a reference architecture is typically chosen at first.

The third step suggests to choose one or more elements of the system to refine. What these architectural elements are depends on which drivers needs be satisfied. Usually it is a module, but it can also be an entire software layer or a component, just a class object, a data entity or a file structure. The right elements have to be selected. They can also be refined, combined, and decomposed.

In the fourth step, the design concept is chosen that satisfies the selected drivers. It is said to be the most difficult decision since the alternatives of design patterns and reference architectures is vast. Usually there are many option with different pros and cons. Cost-Benefit-Analysis can help to make the best selection.

To instantiate architectural elements in the fifth step means to specify the decision of the previous step, that is, to say how the more abstract design concept is applied to the selected driver. To illustrate, how many layer should the layers pattern have or

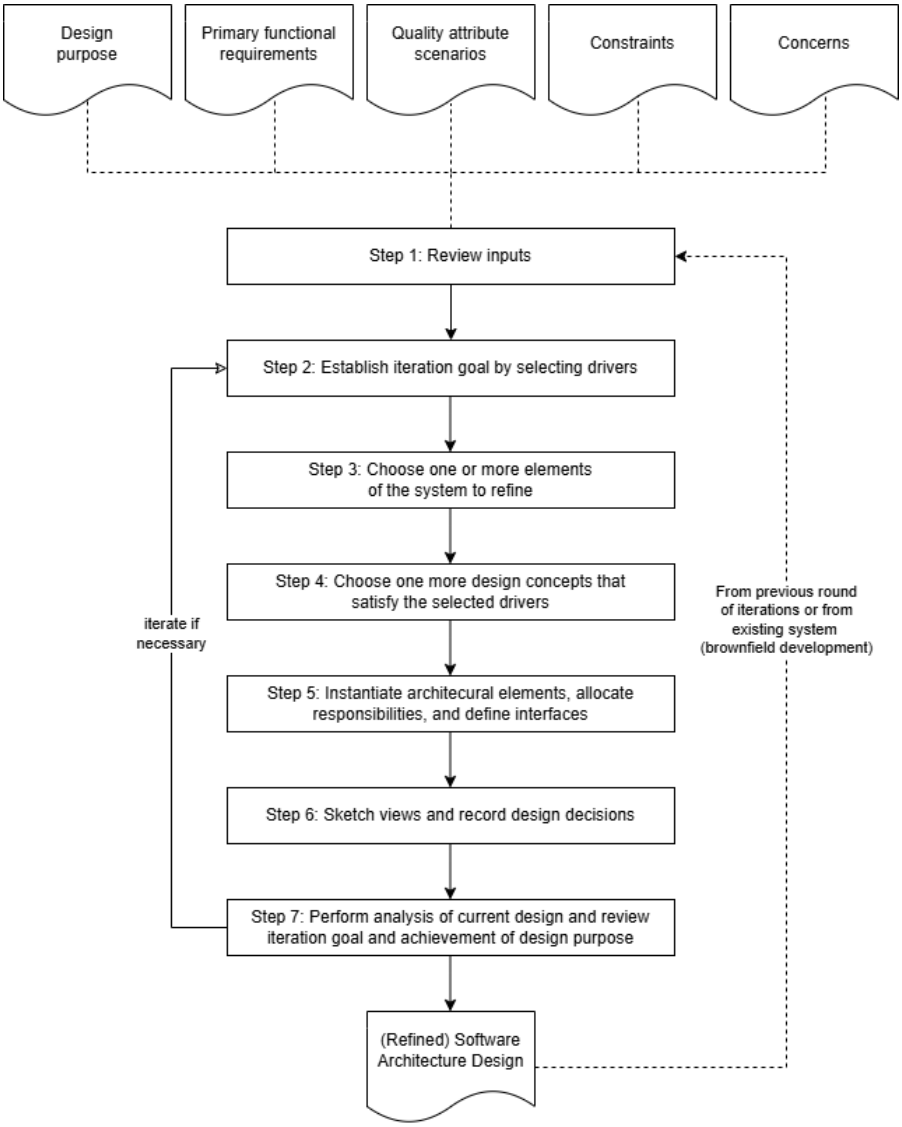


Figure 2.4: Steps and artifacts of ADD (Cervantes and Kazman, 2016: 291)

which of the elements are described in the design pattern. Allocate responsibilities and define interfaces is closely related to such a specification. It must be clear who may access, which part of the data and how the components are connected with each other.

The choice made in fifth is best visualized if sketched down as a view that will be extended and changed in subsequent iterations. The rationale why the design decisions is made should also be recorded. The form of a table is a widely used form of putting the arguments down briefly and clearly arranged. All of this is part of the sixth step in the design process.

In the seventh and last iteration step, an analysis of the current design is performed, the iteration goal reviewed as well as the achievement of the design purpose.

2.4.4 Architecture Evaluation

Justification for Selection

> *Architecture evaluation* is the process of determining the degree to which an architecture is fit for the purpose for which it is intended (Cervantes and Kazman, 2016: 309).

A second dimension to the notion of purpose fulfillment, is the risk of failure. Evaluation reduces the probability of risk that an architecture will not meet the intended purpose or will simply fail on unforeseen issues. Fixing risks identified in the evaluation is a matter of a cost-benefit ratio. Investments of billions, many years of development, and application in savety-critical domains are treated differently than a low budget game. Also the experience that the organization and the architect have in the domain lower the risk of implementing an insufficient architecture. These factors impact the choice of the evaluation method. The architecture tradeoff analysis method (ATAM) is a formalized process that is engaged for large systems in critical domains like defense, finance, or automotive (Clements et al., 2002). ATAM is evaluated by outsiders. Although ATAM is an established method in software engineering for more than two decades, ATAM would overshoot the target of the software project at hand. However, there is a Lightweight Architecture Evaluation (LAE) method by peer review employing the same basic steps, but cutting short on explications and internal information known to the insiders. This method can be carried out within a few hours and seems to be the correct trade-off considering the risk of implementing errors from the architecture, which is more likely when a look from the outside is missing.

Steps in Architecture Evaluation

The first step usually involves to communicate how the evaluation works, that is present each of the steps to the evaluation team. If the participants know the method because they are part of the development team or because the evaluation method is part of the design process, this step can be omitted. Otherwise, the method will be explained and it is made sure that all participants understand the details of the evaluation process. It includes extensive knowledge of the context and the expectations of the stakeholders.

Besides the context of the system, the primary business goals giving rise to the development have to be reviewed as the second step. This can be done as a short system overview from the perspective of the organization and typically addresses the most important functions and constraints (political, economic, managerial, technical), but also the other architectural drivers and involved stakeholders. Special attention is drawn to the recall of architectural significant requirements because those are hardly present in the participants mind.

Third, the architecture and the architectural approaches are reviewed. Since the insiders are supposed to know the system to be developed, a short review of the modules and components-and-connectors are sufficient. Applying all scenarios to the respective views as done in the ATAM is usually not necessary. One or two scenarios are deemed to be enough. In ATAM the review of architectural approaches is taken as an extra step. For LAE it is combined into one, i.e. patterns and tactics that were used in the design process are presented together with their concerns.

Next, step four, the quality attribute utility tree that was produced as an architectural driver in the requirement analysis (section 2.4.3) is presented. A sincere review of all identified quality attributes is done. The tree is updated if new scenarios, response goals, risk assessments, or changed priorities resulted in the discussion. The rather unspecific business goals in the second step may read that the system's portability should be high. Now is the moment for another reality check to see what *high* means and to judge how realistic such specification is likely to turn out, i.e. to which platforms the system is ported and in how much time.

The fifth step, entitled *Brainstorm and Prioritize Scenarios*, only comes into play if a new scenario came up previously and deserves further analysis. As it affects all other scenarios, a new prioritization and risk assessment is instructed.

These prioritized scenarios visualized in the utility tree and ranked highest priority first are then mapped to the architecture in the sixth step called *Analyze Architectural Approaches*. The aim is to link the architectural design decisions and the quality attribute requirements, so that the evaluation team can reconsider the risks, sensitivity points, and tradeoffs. The architect should be able to explain the decisions, but should give enough room for new thoughts.

As a last step, the evaluation team captures all results. In particular new risk themes and sensitivities as well as their discussions. Possibly new tradeoffs, additional scenarios, and tactics are brought to the table, which are recorded in more detail.

2.5 Summary

The explications put forward above, form the theoretical basis prerequisite to what follows in the remaining chapters. Thus these descriptions – as part of the context and domain knowledge – set the frame and direction for the requirements, field analysis, and needs analysis (chapter 3), which upon the architectural decisions and strategies will be based. Together with the requirements analysis, the given context of the RSE and RDM domains in the first part of this chapter are the important ingredients to elaborate the architectural drivers in chapter 4. Although still vague, one can see how RSE services should be placed. As an analogue to research data, research software belongs to the stages of collecting, processing, and analyzing data. They also follow the FAIR principle. These principles set some of the boundaries for the quality requirements, which will have to be mirrored in any incorporation model, whereas functional requirements play a much minor role. Solidly grounded in the literature, in the second part of this chapter, the focus shifts from the context to the methodology for eliciting the requirements, the design process (ADD), and the evaluation (LAE). Justifications for choosing the respective methods are given.

3
Requirement Analysis

This chapter contains the final result of an extensive body of lengthy technical reports and working through them to add to the general context from the last chapter specific use cases, from which the functional requirements can be worked out. The results from both field analysis and needs analysis are used to give the actual state of requirements. Under this premise, reasoning on how research software can be incorporated into existing RDM services, the target state of requirements is explicated on hypothetically derived use cases. For reasons of clear division, the derivation of quality requirements are left to the next chapter.

3.1 Introduction

This chapter starts with a field analysis and a needs analysis. It continues with eliciting requirements from these investigations. The requirements elicitation will be presented separately for RSE and RDM, but the field and needs analysis are showcased conjointly for the RSE and RDM fields because the documented needs in RSE and RDM as well as their environments turned out to be largely overlapping.

The areas of RSE and RDM are perceived as closely related in all of the surveys that have been reviewed. In fact, differences of needs are much more visible among the faculties than between RSE and RDM. A recently published study on a descriptive model of RDM services including research software, for example, reveals high deviations among the humanities faculties (Helling, 2025). From more than 50 surveys considered here ("Surveys," 2025), more than half contain a section *software and data*, which describe the details of software used and research software developed. Additionally, whereas scholars in the humanities and social science state that research software included commercial products general purpose software such as word processors and spread sheets, scholars in science consider a definition that is much more narrow: research software is primarily software highly specific to their thesis including their own developments (Hettrick et al., 2014; Streicher et al., 2025). Hence, the needs in RDM and RSE intermingle to such an extent that it makes little sense to have kept them separate.

The same is true for the field analysis. The guideline for the field analysis are the observable processes taking place in the research data life cycle as introduced in 2.2.2. As expected, these processes look largely similar at the national level and with some minor deviations at the international level. At each stage of the cycle different tools are in operation and this is where the most obvious differences occur. Yet, it is important to note that these differences of needs occur in operating single programs; they are not process-related or even strategic.

3.2 Field Analysis

RDM usually starts with planning the data management needed in a research project. The research proposal and RDM planning goes hand in hand. Really the later is part of the former since it includes specifying the requirements for software, storage size, data analysis and curation, long-term availability and accessibility strategies of the data, the costs, and many more. Data ethics and legal requirements are particularly tricky. All of this must also be described in any research proposal. Briefly put, to offer help on this opaque terrain, all major RDM institutions offer elaborated consulting. Moreover, software, mostly in the form of a guided questionnaire, is provided that points the researcher to the typical pitfalls of data management learned from the many painful mistakes of the past. Although such data management plans can also be generated online ("DMPOnline," 2025; "RDMO," 2025a), the better alternative is to have the institution maintaining an instance, in which DMP data are stored and can be revisited without restrictions or unexpected future changes in the licensing. At the time of writing, about 60 German and Austrian research institutions decided to run their own DMP tool ("RDMO," 2025b). This comes with another potential advantage that would play a role in designing software architectures for RDM and RSE services. Data stewards working in the field have more recently discussed the idea of using the data entered in a DMP directly for preparing a research project at the operational level by the computing center. Such an approach works best with an existing single-sign-on (SSO) infrastructure because once the funding agency emitted a positive decision, the departments responsible for supplying storage, virtual machines, software, and the like may access the specifications in the DMP. Also the DMP credentials are the same as for accessing all the provided infrastructure by researcher. This could become even more simpler if machine-actionable DMP are fully implemented and usable. National infrastructural projects such as *DMP4NFDI* ("NFDI," 2025a) are under way that promote this idea.

While the above is a useful add on that would speed up organizational delays, other infrastructural requirements are must-haves as one follows along in the research data life cycle. The stages *collect & create* as well as *analyze & collaborate* is especially relevant to RSE. At this phase of a project, software engineers (if required) are engaged to design programs and data models. Others carry out pilot studies if not already done as preliminary work or start right away with the main study and data collection. Nearly every research project is based on data that is sooner or later digitized. Within the process of data collection, data analysis, and overall the work with the data during a research project, essentially all data centers offer virtualization technologies with differing degrees of customizable GPUs and storage, webengines and database systems. However, at national and international levels qualitative differences exist that can best be summarized from a low to high range of services and hardware. While a dynamic

website on centrally maintained servers with little choice on a specific technology is standard, orchestrated computing clusters for high performance computing is restricted to very few institutions specialized on compute-intensive processes of large amounts of data (local and national research centers such as Helmholtz, Max-Planck, Academy of Science, CLICCS). The mainstream strategy of administered data centers, however, is to make available a cloud solution with tools for office work and moderate storage that can also be used for data exchange, a choice of two or three technologies of a centrally administered relational database server, a web space inclusive two or three web server technologies, and an independent code repository, usually *gitlab*, that often contains automated deployment with adjustable pipelining and build processes. Less common are centralized communication platforms (*Mattermost*, *Mastodon*), wikis (*Confluence*), project management and ticketing (*Jira*), coding (*Jupyter*) or a centralized containerization solution (*Docker*). Again, there are nationally funded projects that foster more effort in this direction. *Jupyter4NFDI*, for example, addresses a more efficient use of computational resources by allowing researchers of distributed *Jupyter* hubs of all participating institutions to be used commonly and independent from the programming language or installed dependencies ("NFDI," 2025b). *NFDI.software* strives to create a central market place for research software to enable more reuse and sustainability, but also findability and accessibility will be improved by orchestrating the research software landscape.

Going further on in the research data life cycle, the stages of archiving, sharing, disseminating, accessing and reusing come into play. Data repositories are the all-in-one solution for the rest of the cycle. The publication of research data in a repository usually takes place at the end of a project at a time when the data is unlikely to change although this is not a necessity. Open source repository software that covers the vast majority of all international educational and public organizations are *Dataverse*, *DSpace*, *EPrints*, *Fedora*, *Invenio*, *Islandora*, and *Samvera* (Greer Klein et al., 2024). Some of them come in different flavors of view layers with respect to the technology (*React*, *Angular*). Every solution can be configured to a limited degree and by changing the code base to whatever the institution requires. The later is rarely done for reasons of maintaining versions and updates. Configuration covers the storage infrastructure, SSO, layout and design, access of funding agencies, or PID generation.

As a last step in the field analysis, the involved stakeholders of RSE and RDM will be considered. Principal investigators (PI) and research staff certainly belong to the inner circle of direct users and profiteers. Collecting their needs is essential for primary functionalities and qualities. However, it is also clear that this group is seldom aware of conflicting target relations of qualities, e.g. high degrees of performance and security. Second, it is challenging to find out what is really needed behind the wish list, i.e. to map the wish to the real need to a specific functional requirement. In addition, the interests between researcher and principal investigator as well as among the team of researchers must not be in sync. On the same analytic layer, the technical staff that will have to maintain, implement, and adjust the software architecture. The above surveys are rather unspecific about the level of qualification of the available staff present at the data centers and libraries that are in charge of this task. Yet, it is uncommon to outsource larger software projects to private companies. Third party funding and collaboration is the established way of getting a specific open source and community software implemented. Hence, funding agencies also belong to the group of stakeholders. Since they are not directly involved in the operational part of a project, they are part of the outer circle.

Their interest consists in improving the societal situation and – on a smaller scale – the community. Indeed, the financial situation and the involved costs are important, too. On the same level of influence also act stakeholders as community gate keepers and the higher hierarchical layers of the institution the project is implemented, i.e. team leader, dean, and university chancellor. Competitors offering similar commercial products or services from whom best practices or published reference architectures could be used would still be stakeholders for a public research organization for their specific knowledge and interest in potentially innovative products. Yet those receive little consideration.

3.3 Needs Analysis

The above field analysis reveals what is already prevalent typically. It also points out who the stakeholders are. The needs analysis is now supposed to disclose what the stakeholders actually want whereas the inner circle of stakeholders provides the most precise indicators for functionality. Together with the surrounding environment, it is possible to detect gaps in the supply of services. So, the following needs analysis focuses on the first layer identified in the section above principal investigators, research staff, and technical staff of the data and RDM centers.

Although representativity of the results is not necessarily applicable to the international or national situation, a number of locally collected surveys limited to a few institutes exist,[1] which reveal a more differentiated picture of the motives and needs (Streicher et al., 2025). Due to the qualitative nature of these interviews, it is still possible to cross-relate the use cases and identify common issues and needs.

The first stakeholder group to consider are the principal investigators and the researchers. While in smaller projects these two roles coincide, from an analytical point of view, it makes sense to strictly separate them. The principal investigator usually wrote the proposal of the research project and went through some bureaucratic hurdles to be successful. So at project kick-off this person not only has budgetary sovereignty of the project cost and the responsibility to achieve the project goals, but also carries the burden of the main organization including recruitment of capable staff and other resources such as the technical equipment.

From this background, PI interests' consist in minimizing the bureaucratic overhead. As an ideal, they would like to specify everything needed to carry out the project only once, e.g. in the DMP or funding proposal and have all the material, software and hardware available at the project start and thus reduce any additional requests and applications for storage, web-spaces, CPUs, computing clusters, or the like, but expressly excluding the research staff. The same is true for reporting obligations. The main focus lies in strategic effectiveness of the decisions, i.e., is the staff capable, are the methods and resources the right ones to achieve the objectives of the proposal?

The researchers give much more attention to the efficiency to carry out the methodological work of a project. One can observe a wide gap in terms of IT-skills and RSE-know-how across the faculties. But also within a faculty the gaps exist regardless of the very homogeneously distributed high quality of the staffs' educational qualification. In general, all prefer high degrees of automation or at least the possibility for it. Researchers

1. See Wörner (2015) as an example for the case at the University of Hamburg. Similar results were put forward at RDA conferences, e.g. in 2015 for social and natural sciences and FORGE in the years 2015 and 2016 for the humanities or internationally (Katz et al., 2021).

without prior IT-experience ask for intensive consultancy when setting up the technical environment ranging from deployment pipelines to data modeling. Especially the later is likely to change as research projects progress and unforeseen variables emerge. The need to be addressed here is a compensation for the missing knowledge of modeling techniques and understanding data models. Researchers with a decent technological background would still express their concerns with keeping software libraries and its dependencies up-to-date. This is especially true for the fit of operating systems on centrally managed web-spaces, database servers on the one side and the research software on the other side. Yet, discomfort is also reported for self-managed virtual machines if the data center asks for urgent updates due to security reasons.

A wish that is put forward by advanced staff is a centralized containerization infrastructure, in which deprecated software could run, knowing fairly well that such a solution could also fail in the long run, but rather unlikely during the time of a five year project. A need specifically addressed to RDM is an improvement of the visualization of research data when published in some repository. A typical example applies to data from relational databases whose relations are condensed into tables in text formats for reasons of sustainability. The sustainability issue of software is indeed seen as one of the major future issues to be solved in RSE most urgently (Blech et al., 2022).

As far as the technical staff is concerned, data curators have a high interest in a smooth data transfer as the project comes to an end. However, as conformably reported in a wealth of use cases, it corresponds rarely to reality. Unknown standards and formats, inconsistent data resulting from unthought-out changes in the data model, poor documentation, or missing data are on the daily work schedule of a majority of data stewards. So the need uttered here is to receive well documented and consistent data. When asked for the cause, the answer vary a lot, but there seems to be consensus in the data curation community to improve and intensify communication during the projects. Relevant changes should be notified in due course.

Few technical staff taking care of the hardware indicated that the provided computing power as well as storage is seldom fully utilized. They would prefer a more concise tuning of resources because there is too much overhead in the unused resources that lay idle. The logic of this need is comprehensible and for the relevant part in line with the PIs' need to reduce the bureaucratic burden of applying for additional resources if required. As a consequence, the practice among PIs is to apply for the maximum right away.

As the analysis reveals, the needs of one stakeholder may stay in contrast to another. They may also compensate or complement each other. So the research staff may need full-time support and consultancy for IT services, but it contrasts to the number of projects in relation to the number of staff employed. On the one hand, the executive of the RDM center will not be able to allocate sufficient resources to each of the projects and therefore will suggest compromises in the support so that in relation to the project size work force will be allocated equally. On the other hand, the more the technical staff knows about the domain of the research project, the better the support will be.

3.4 Requirements Elicitation

The aim of this section on requirements elicitation is to translate the above needs into concrete requirements while considering the field of RDM and RSE. The tasks and actions on the four groups of stakeholders, principal investigators, researcher, data

stewards, and technical staff, will be reduced to an extent that seems plausibly useful for devising a software architecture that incorporates RSE processes and software into RDM. As proposed in requirements elicitation, needs stated by stakeholders are the most valuable source of information in finding relevant requirements, but the real needs are hardly known by the stakeholder. It is experience, further reasoning, and above all, the usage of the software product by the stakeholders, which transform reported needs into the needs actually intended. This will be done in two steps. First, the actual state of requirements is displayed for the relevant stakeholders. Second, from the actual state of requirements a target state of requirements is derived engaging domain knowledge as well as reasoning. The logic behind step two is that the interplay of independent stakeholder evokes new or restricts prevalent requirements. The form of depiction are use case diagrams since it is the agreed standard in architectural design.

3.4.1 Actual State of Requirements

The researcher in the role of the research software engineer creates a functioning research application. Usually such application stores its research data in a data base whose data model is also designed by the researcher. Depending on his or her experience, outside expertise from professional software engineers or data modelers are requested; often it is limited to consulting the RDM staff. Moreover, the research software engineer sets up the deployment pipeline, for which further communication with the technical staff of the computing center that administers the software repository is needed. The researcher uses the pipeline to efficiently run new versions of the research software. Especially for collaborative work web engines come into play that require regular software updates. This also applies to research software that typically draw on external libraries. And again, it depends on the level of dependency that the researcher or the project owner decides to have. If little services that are centrally provided are used, the degree of independence is higher and so is the work for maintenance and the technical knowledge.

The researcher may also take on other roles, that is, the role of a data collector or the role of a data analyst. In this case, the researcher is seen as the user of the application and is much more in line with the understanding of traditional data management. The focus of these tasks comprises rather the consistent data input, elaborated documentation, use of standards, and independent data formats. In the use cases depicted in figure 3.1, the researcher is both software engineer and data analyst. Yet, data analysis is limited to only one task. The RSE tasks are the main focus.

Although, the role of a principal investigator or a project owner is yet another role of a researcher, the importance of the involved tasks seem to justify an extra contemplation. In figure 3.2, the use case of the managerial tasks in a research project is shown. The first task to note is planning and delegating. The project owner follows a plan of activities and tracks the fulfillment of all other tasks that contribute to the project's goals. Indeed all this needs to be communicated to the researchers. In any larger project, the principal investigator will recruit other researchers who will be allocated to the research tasks. This also reveals the hierarchy of commands where the PI *commands* the other (recruited) researchers. Other tasks such as the budgetary management and tracking goal achievement follow from the responsibility of leading. It also includes the sovereignty of which research outputs can when be published. Procurement of all necessary equipment, workplaces, hardware, software, and the like

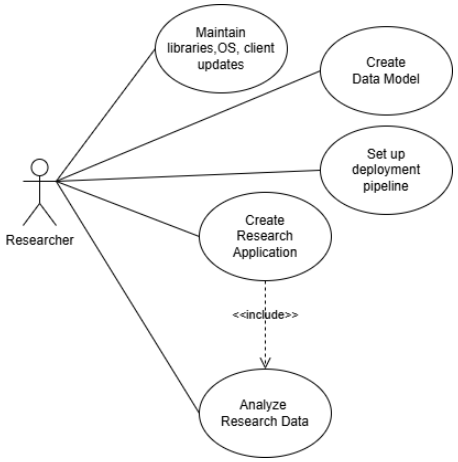


Figure 3.1: Relevant task in RSE and RDM

is often delegated as a task to some technical staff, yet the responsibility of provision still remains at the realm of the project owner.

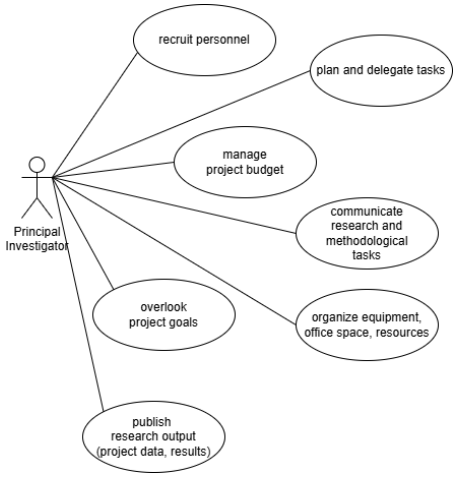


Figure 3.2: Managerial tasks in research projects

The role of the data curator is on the service side. It is supposed to directly support research activity when it comes to managing their data. Looking at figure 3.3 and comparing it with figure 3.1, it becomes apparent that some of the tasks intersect. For example, *create data model* as a researcher's task and *correct data model* as a curator's task suggest that both parties are involved in data modeling – incorporating data consistency, data formats, and standards – all of which the research application has the most direct impact. In other words, the more conscientiously the task of creating a mature data model is done, the less comprehensive the core tasks of the data curator – format, standardize, and manage inconsistent data – will be. The same is indeed true for documenting. On the one hand, blurry boundaries of task responsibilities bear the risk of carrying out the same task multiple times or not at all. On the other hand, efficient work sharing bears the chance of synergy effects. From the point of architectural design, one should concentrate on efficient work sharing and reflect this idea into the architecture.

Finally, a rather abbreviated look is taken at the task of the technical staff of a data center. This role is on the service side as well and supports research activities. Again, there are intersections with the researcher and it is easier to separate the

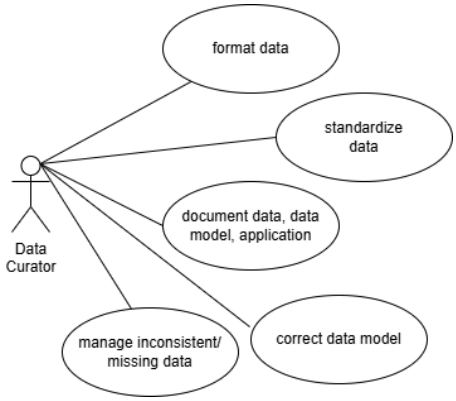


Figure 3.3: Relevant tasks data curator

scope and responsibilities from the data curator as from the researcher. As figure 3.4 delineates, the intersecting tasks depend on whether the researcher chooses to use central services on the cost of flexibility because then the task of maintaining a data base server or an operating system is done by the technical staff. Also most of the tedious configuration work can then be delegated to the computing center. The actual procurement of hardware, its set up, and operation is part of the job of technical staff only. Security measures and certification, however, is often well-defined shared work whereas the researcher takes responsibilities on the last three levels (application, presentation, session) and the technical staff at any level below that (transport, network, data link, physical) according to the OSI reference model ("ISO," 1994).

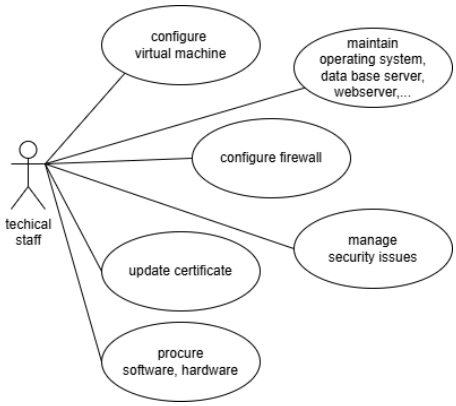


Figure 3.4: Relevant tasks technical staff

3.4.2 Target State of Requirements

Incorporating research software into an RDM workflow implies as a first step in architectural thinking that the relevant tasks of the involved stakeholders and their possible intersections have to be considered. One should also keep in mind that the objectives of the stakeholders may be in conflict with each other. Reasons of diverging objectives are sought in the nature of the task or in the expectation of what a task should comprise. To illustrate the conflict that the nature of the task provokes, one could think of the principal investigator and the technical staff of a computing center. The principal investigator who applies for a maximum of resources to save the bureaucratic overload

directly conflicts with the technical staff who observes that most of the granted resources lay idle and cost maintenance time. Similar conflicts are reported for the expectation of a task. Between the researcher and the data curator a conflict is commonly reported that emerges on the question to which extent research data has to be consistent or which degree of detail should a documentation contain. Resolving conflicting objectives means that independent requirements have to be brought in line. And this implies compromises in the architecture.

At this stage of design, the tasks and requirements documented in figures 3.1, 3.2, 3.3, and 3.4 as use cases have to be *re-thought* in terms of a new ensemble, an interplay of resolved conflicts. The result of this process is shown in figures 3.5 and 3.6, whereas the the later extends the former by the technical staff. This is done for completeness and shows an additional possibility of extending the incorporation to the service of a computing center.

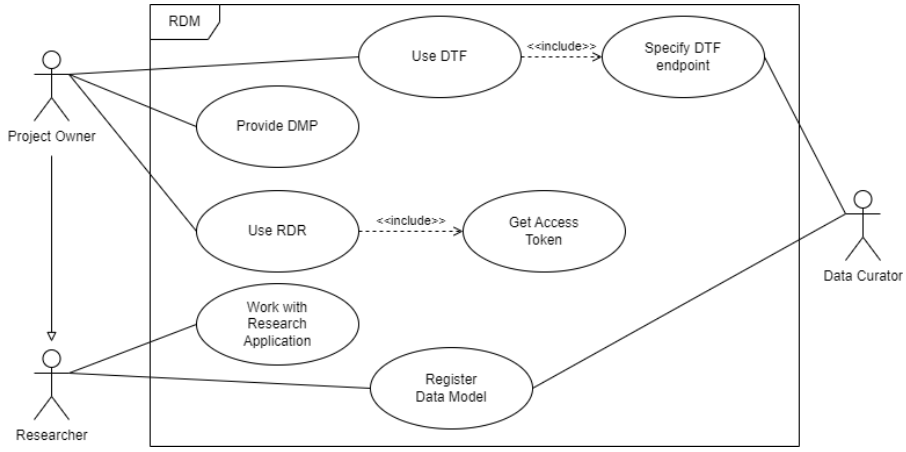


Figure 3.5: The requirements of researcher in RDM.

To begin with, there is an assumption to be made about an architectural component used in figures 3.5 and 3.6. Conceptually, it is as of now an abstract unit, which is not yet specified in any way, but from which it is clear that it will contain some kind of interface between research and RDM. The only thing that can be assumed about such a component is that it is supposed to facilitate the data transfer from the research application to the RDM infrastructure. Therefore, it is simply called *Data Transfer Facilitator* or short *DTF*. The existence of DTF is mandatory and a first result of the requirement elicitation since from the previous section follows that data will be exchanged between the researcher and the data curator.

It is this unit that will put the metadata and, in the progress of the project, the research data in the correct format and standard so that the users will not be asked to re-enter anything that is already available in their research software or in the DMP whose information should also be automatically retrievable via another interface definition.

The use case in figure 3.5 differentiates between the role of a project owner (Principal Investigator of a project) and the role of a researcher. Indeed, the depiction includes the possibility that the project owner can also fulfill the role of a researcher. Yet, it must be clear that the project owner has the access rights to the infrastructure for the overall project data. It follows from the responsibilities given in the use cases above (figure 3.2). Therefore, it is the credentials of the project owner to which the interface definitions have to be linked. This is the case for both the DMP, in which the attributes of the project's

data management plan are kept, and the repository harboring the metadata as well as the research data. The project owner will also determine the time at which the research data could be transferred to the permanent storage infrastructure, which is done via the DTF. The specific endpoint definition is defined by the data curator based on the information received from the data modeler who usually is one of the researchers. The role of the researcher is to create the data models and work with the research application, i.e. entering research data or analyzing it. However, the researcher should not interfere with the project owner's scope of responsibility and is therefore not eligible to define access tokens in the repository since it would enable the researcher to change access rights, licenses, or embargoed data for which the project owner is legally held accountable.

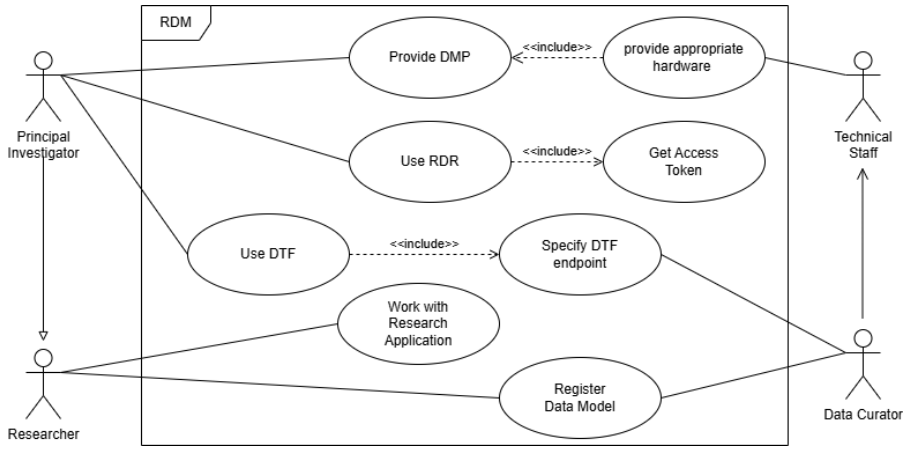


Figure 3.6: Requirements of all relevant stakeholders in RDM

Figure 3.6 extends figure 3.5 by the technical staff of a computing center. At this stage of development, it is not clear if the additional use case of *providing appropriate hardware* by the technical staff will be harbored in the architectural design process because these services traditionally do not belong to the offerings of RDM. However, as the field analysis revealed, in some institutions RDM and computing center merge together and so are the roles not strictly separated as the logic of the use cases suggest. Besides of this, the use cases of the technical staff would diminish the bureaucratic overload of the principal investigator as it came out in the needs analysis. Once the technical staff has access to the DMP and gets the message of the project's acceptance, the procurement could start. The same could be done for providing office space, but it would involve even more administrative departments.

3.5 Conclusions

There is now a clearer picture of how an integration of research software in the RDM services could take place from a conceptual point of view. Also the use cases give a much more detailed idea of the necessary quality attributes that will be spelled out in the chapter 4 on architectural drivers.

Since the analysis revealed that tasks of different roles intersect, it gives the software analyst the chance to clarify the responsibilities more specifically and reflect these clarifications in the software design. Although two sides, research (at the left) and data service (at the right in figures 3.5 and 3.6, can be distinguished, the tasks carried

out at each side cannot unequivocally allocated to research or data services. The research software engineer deals with a lot of methodological work belonging to data services. Sophisticated data curation requires extensive domain knowledge of the research discipline and at best a solid understanding of the research carried out as well. Yet, the solution is to define the task independent of the two sides of research and service work in close coordination with the parties involved.

By the same token, the analysis points to a conflict of objectives between the requirements of stakeholders, i.e. institution versus researcher. Again, it is paramount to identify the conflicts so that a solution can be found. The target states of requirements suggest that it is possible to find an optimal point of compromise if cost efficiency of the conflicting parties is the guiding principle. So task intersection and conflicting objectives between researcher and data curator is translated into the DTF software, which requires an endpoint definition by the curator and a registration by the researcher or research software engineer respectively. Once the endpoint is installed, data transfer is a matter of logging in both the RDR and DTF, getting a token from the RDR and confirming transfer in the DTF. The project owner is encouraged to commit snapshots of preliminary data whenever a major milestone is reached. So, a history of the development of the research output in several versions is available and subsumed under a main DOI. Indeed, it is citable at conferences and talks as the project goes on. The crucial point in this approach would be that changes in the data model that are not covered by the endpoint definition could be checked by automated tests in regular time intervals. If incompatibilities were discovered, the project owner would be informed and asked for an alternative endpoint definition that would fit the needs of the changed data model. The advantage here is that changes are corrected in due course when the implicit knowledge of the change can still be recovered as if it were just another function of the design pattern of the original application. It is not postponed towards the end of the project, for which experience showed that it is difficult to re-establish the missing information. In other words, promptly keeping track of the changes in research design, data storage, and implicit knowledge of staff is encouraged and the risk of losing information is minimized.

Table 3.1 puts the analysis from the above analysis into a condensed form to have it ready for the selection of primaries and subsequent architectural design. [2]

3.6 Summary

This chapter was dedicated to requirement analysis and its outcome answers the first research question of the thesis at hand. It is the point of departure for all remaining analytical work on quality attributes and a final architecture proposal. Based on a field analysis of RDM and a needs analysis of relevant stakeholders in the beginning of this chapter as well as the domain knowledge from the previous chapter, the requirement elicitation shed light on the state of use cases. From there a target requirement scenario could be devised for which the architecture can then be designed. The final deductions made in the conclusion section suggest that the specification in a software component helps to diminish tasks intersections and conflicting objectives in the traditional workflows of RDM services.

2. Some of the wording in table 3.1 is taken verbatim from the three case studies in Cervantes and Kazman (2016).

Use Case	Description
UC-1: Detect faults	Periodically the system contacts the servers via the endpoint definition to see if they are working.
UC-2: Display information	Working servers are matched to project names. Endpoint definitions and login credentials for data base servers (read rights only).
UC-3: DMP data	DMP API requests information relevant to hardware or project meta data.
UC-4: RDR transfer	RDR API specification is generated for meta data and research data
UC-5: Log in	A user logs into the system through a login/password screen. Upon successful login, the user is presented with different options according to their role: PI is presented with the options to enter RDR access token, RDR data transfer, data viewing, and data download. Data curator is presented with the option to confirm endpoint, view monitoring, and role management. Researcher is presented with options to register or update a data model. Technical staff is presented with a list of hardware requirements.
UC-6: configure endpoints	data curator tests and adjusts endpoint definitions.

Table 3.1: Description of identified use cases

4
Architectural Drivers

This chapter elaborates on the architectural drivers serving as a prerequisite to the subsequent design phase. As a main focus the quality attributes are comprehensibly developed and concrete response measures for them are provided before deriving the primary functionalities based on a short risk evaluation. Indeed, design purpose, constraints, and architectural concerns are also described and its relevance is made plausible.

4.1 Introduction

For reasons of a clear arrangement of the structure, this is a separate chapter on the five architectural drivers that are part of the architectural design process of ADD to be described in the next chapter. As depicted in figure 2.4, the design purpose, quality attributes, primary functionalities, constraints, and concerns are presented. While all of them play their part in the design process, the quality attributes are of particular significance since they address the second research question of this thesis. Hence, the main message here is to give a more detailed account on the quality attributes and keep the other drivers as brief descriptions that follow from the previous analysis. The functional requirements from the previous chapter are only briefly mentioned here for completeness to appropriately reason about primary functionalities. They have been treated in depth as a result of the first research question. The general theory on architectural drivers are elaborated in 2.4.3 and will not be repeated here. The focus is now on the specifics of an architecture that helps to integrate scientific software into the research data management infrastructure.

4.2 Design Purpose

The business goal of this software project can be paraphrased as the main goal of this thesis. It is to flexibly interconnect diverse research software applications to the RDM infrastructure. The requirement analysis increasingly reveals that such an interconnection or incorporation is best achieved on the basis of the research data that

has to be transferred to the RDM infrastructure. Sooner or later such a transfer is made and it seems that an early transmission is more advantageous if the data can still be used, extended, changed. Thus, the design purpose is to ease the transfer of data produced by any number of research software applications whereas the transfer should be as comfortable and efficient as possible for all stakeholders directly involved.

Efficiency problems of a wealth of data transfer scenarios and case studies ranging from long forgotten to active data applications is constantly and vividly reported in the literature as a major issue in data curation (Altman and Landau, 2024; Kouper and Stahlman, 2025), but as of now there is no proposal yet made or a working solution implemented that could be used as a reference. It can therefore be stated that the software would be implemented as a greenfield system in a relatively novel domain. However, the domain has several established software systems for data repositories, which have to be considered in the architecture.

Since it is a greenfield system for which no concrete quantization for the definition of quality attribute scenarios are available, an additional design purpose is prototyping of some subsystems or processes. It is not a primary design purpose, but such a partial prototype is advantageous to quickly react to changes in requirements. And those may emerge no matter how careful the requirements analysis has been carried out. Once the present architecture is going to be put into a productive application within an approved project, feedback loops from requirements to the quality attribute may happen and this may also have repercussions to the architecture (see figure 2.4).

4.3 Quality Attribute Scenarios

According to ISO/IEC FCD 25010 Product Quality Standard ("ISO," 2023; Bass et al., 2022: 210) the most common software quality attributes are grouped into eight categories. Not all of the common notions are listed there that are frequently encountered in the literature, e.g. extensibility in Cervantes and Kazman (2016: 110). So it is treated here as a subcategory of maintainability, to which modifiability is a subcategory as well. Also, there are some typical phrases, which seem to be repeatedly used in architectural description of quality attributes. The wording of quality attribute descriptions that happen to be similar to the ones developed here are adapted from Cervantes and Kazman (2016) and Bass et al. (2022).

From the needs analysis of the stakeholders follows that usability and modifiability are especially important whereas deployability and integrability follow from the field analysis and the work efficiency claim of data curators. Still, these analysis hardly provides any quantitative measures needed for specifying quality attributes. To get as many realistic figures as possible for a limited number of researchers and projects, a prototype that comprises only some of the functionality was developed (Peukert et al., 2025).[1] In this way, measures for deployability, integrability and to a more limited extent to maintainability could be found. For usability attributes, however, the prototype does not seem reliable enough.

1. https://gitlab.rrz.uni-hamburg.de/softwaretools/heurist-sapi.git

4.3.1 Usability

The usability attributes are derived from the need of the user groups not to invest additional time in finding out how to operate a steadily growing number of software programs, which suggest an intuitive understanding what needs to be done in the software is paramount. In other learnability use cases of an equally moderate complexity of an web interface, few minutes of experimentation have been proposed (Bass et al., 2022: 199) so that a measure of five minutes seemed justified for the present case. Time constraints are also satisfied if the user is quickly notified about the success or failure of a change. This should be done as quickly as possible. Since performance measure are only indirectly relevant – they have not been mentioned for this context, it suffices to give feedback on the what is happening in due course. The number of 10 seconds for feedback is somewhat arbitrary for missing any other figure, but it is plausible when considering general browsing behavior.

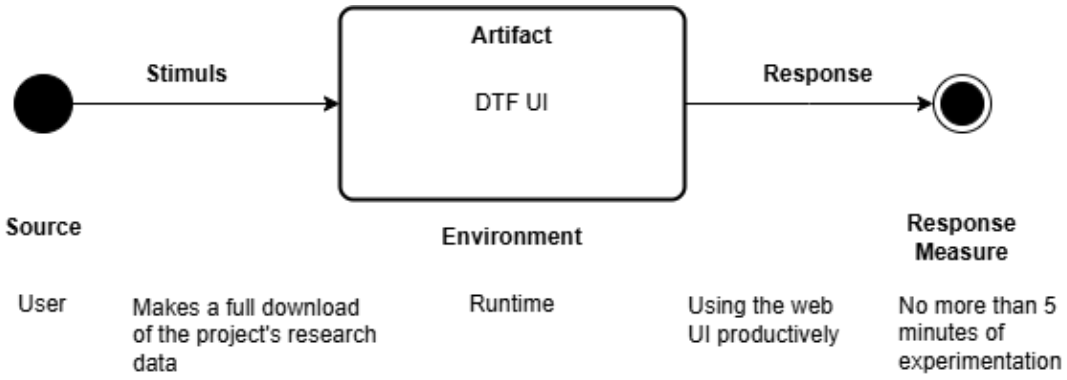


Figure 4.1: Usability Quality Attribute Scenario

Figure 4.1 shows the attribute scenario *QA-1*[2] for a user in the role of a researcher that first logged in the web interface of the DTF component. The goal is to design the front end in an environment that is as clearly arranged as possible so that the researcher would not need more than five minutes to learn how to make a data download. The same response measure will be assumed if the role of the user is the principal investigator who would make a data transfer to the repository (stimulus) whereas this measure does not include the time that the PI would require to get the access token from the repository (QA-2). Although this process could also be automated or made optional in a follow-up version of the DTF, for legal reasons – i.e. directly confirming consent to the terms and conditions of the repository software as well as saving the token at different place than the repository itself – the first suggestion of the DTF architecture will keep apart data transfer and reception of a token.

A third usability scenario specifies the feedback that a user should get (QA-3). As a source the user has the role of a researcher that configures a new endpoint for data retrieval taking place at runtime while using the DTF UI in production. The response measure here is to get a notification within less than ten seconds if the configuration is working.

2. As a short reference and in accordance with best practices in software engineering, the quality attributes will be abbreviated QA plus a running number. These IDs will be referred to later in the text.

4.3.2 Modifiability

For more complex cases of an endpoint definition, the data curator should have the possibility to implement the endpoint manually in the source code. This may be the case if a new technology of a database server is employed (see 4.3.3) or the complexity of the query to get all results is so complex that it needs several requests that are concatenated to the final data set. Figure 4.2 reveals the scenario (QA-4) for manually setting up a data transfer endpoint. The response measure of less than five hours of development time derives from the frequently encountered need of researchers to have a change done within a working day. Adding sufficient person hours for deployment (see 4.3.4), five hours is close to the maximum value possible for a reaction within one working day. With the help of a prototype and a real humanities data transfer project, a process time of about three hours including communication with the researcher was found. However, the prototype is a very simplified version of the real transfer component and, additionally, only little time needed to be invested in testing so that the estimation of five hours is deemed to be a realistic figure.

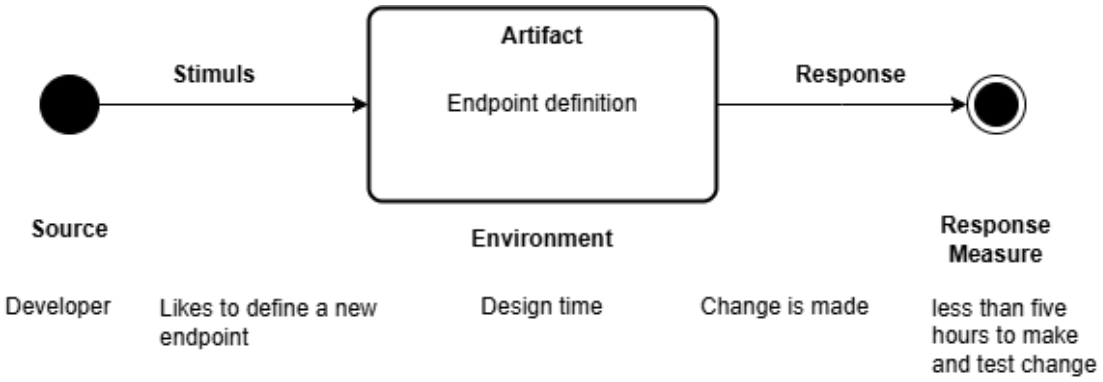


Figure 4.2: Modifiability Quality Attribute Scenario

A second maintainability scenario (QA-5) circumscribes the standard procedure of adding data transfer specifications. This scenario parallels the third usability quality requirement (in section 4.3.1) described above, in which the endpoint definition is configured by specifying the database access credentials and a working query at runtime. The attribute refers to the extensibility of the system and thus shall not interrupt production or necessitate new deployment while the new configuration takes effect. The measure here is that the system has no downtime at all for making this change.

4.3.3 Integrability

Closely related to the modifiability attribute is the attribute of integrability. There are, again, two scenarios that are predicted to be important for the final operation of the system. It is likely that new technologies will have to be integrated into the systems architecture in the upcoming years or that the existing components will be updated or changed completely. In anticipation of these developments, the integrability quality attributes were defined. The driving force of including integrability is the experience of steady change made of data curators in the past as was explicated in the needs analysis.

Figure 4.3 reifies a probable (though hypothetical) scenario of integrability (QA-6). The developer (source) likes to make available a new database component in the DTF

system. For this purpose one person month of effort are estimated. This measure has been tested with a real data project that intended to include *CouchDB* whereas half the time was spent to establish several test scenarios.

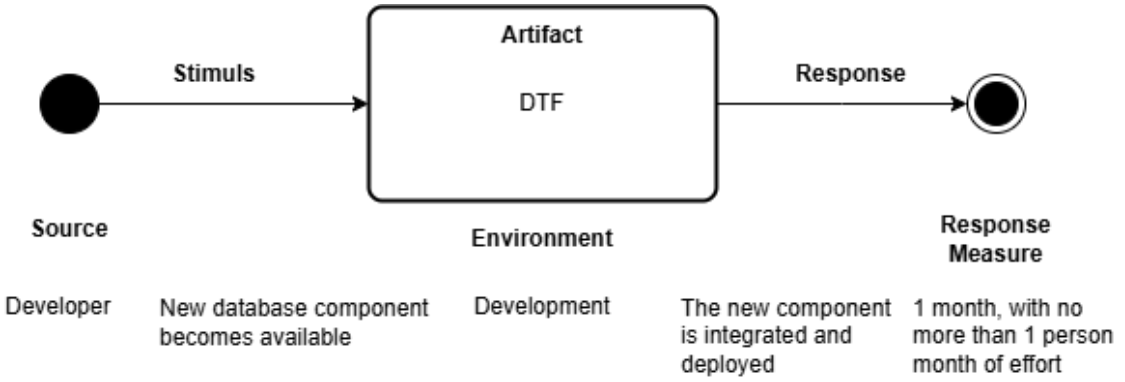


Figure 4.3: Integrability Quality Attribute Scenario

The same response measure is estimated for adjusting the existing components (QA-7). There are no concrete data, which this measure can be based on, but because other software components in the data life cycle such as DMP and RDR are connected by interfaces – and there are no indicators that interfaces are missing in future components – the integration is limited to the mere adjustment in the standard interface definitions. Thus workload is generously estimated. A likely time surplus could be spent for testing.

4.3.4 Deployability

The scenario in section 4.3.3 (QA-6) visualizes a developer integrating new components or making modifications to the source code of the production system. Such changes require the system to be deployed. For these in general and the presumably more frequent scenario of modifying a complex endpoint in particular (QA-4 section 4.3.2), figure 4.4 depicts the follow-up scenario (QA-8), which states that a new deployment is desired within a day. This measure has been intensively tested with the conditions prevalent at a local university data center. The person hours include the effort attributable to the technical staff at the data center for a worst case. Although the response measure may be a specific local phenomenon and may not be representative at the national or even international level, it still gives a possible limit value, which has to be treated as a bottleneck. And it is this environment, in which the software will be operationalized first. Deployments carried out in a more competitive field may be faster. Nevertheless, the stated requirement can be fulfilled if the reaction time is 24 hours corresponding to the need “within a day” (10 hours deployment plus 5 hours developing time).

The second deployability attribute (QA-9) goes back to the data curators and technical staff alike. The system deployment procedure shall be fully automated and support a development, a test, and a production environment. This response measure is a *have* or *have-not* decision. It has proven to be a realistic procedure at the local level in about a hundred or so research projects. Feasibility also has been confirmed at the national level. It is a standard in the relevant literature (e.g.Cervantes and Kazman, 2016: 110).

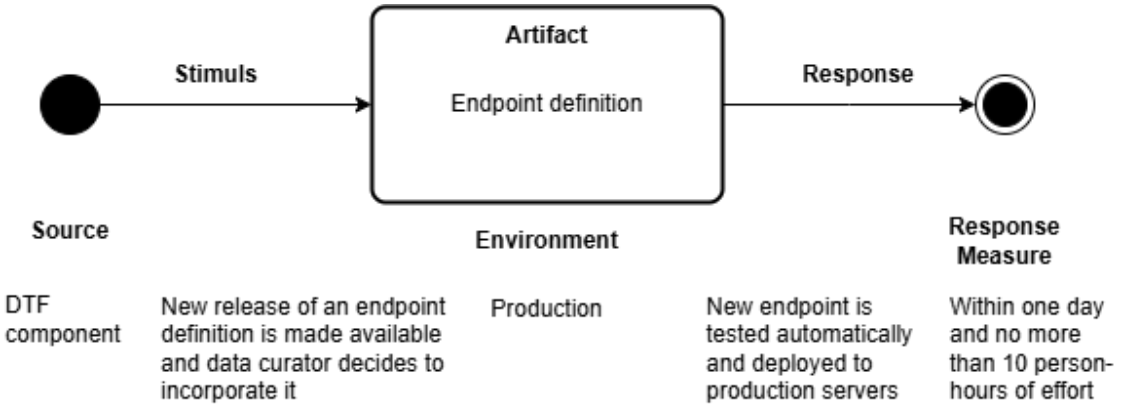


Figure 4.4: Deployability Quality Attribute Scenario

4.3.5 Utility Tree

To be applicable at the more general national level, the requirements have not been collected from the internal stakeholders on site in a quality attribute workshop (QAW) as recommended in the literature, but derived from collective reports and published documents that are somewhat more representative for the overall situation in Germany (and in part at the international level). Still, a prioritization of the attributes is necessary to organize the architect's thoughts and to have a non-arbitrary selection criteria if attributes happen to be in conflict.

The literature recommends to heed two dimensions: the business risk and the technical risk (Cervantes and Kazman, 2016: 24; Bass et al., 2022: 284). Following these guidelines, the technical risk encodes the difficulty that may arise when implementing the software specification, i.e. the risk that the quality attribute would fail to be achieved. This estimation can be given by the architect whereas the more experienced the architect with projects in a domain is, the more reliable the estimation will be. To avoid too much of a bias of a single architect, the risk estimation is crosschecked with the results of the field analysis for any contradictions.

The business risk aims at the customer side and how relevant the quality attribute is for the user's significant needs, which translates here into the perceived stakeholder's importance. This evaluation is based on the needs analysis, but is much more subjective than judging the business risk. The result is depicted in table 4.1, but also part of the utility tree in figure 4.5.

The risk estimation, quality attribute, and response measure are now put in a more condensed form to work with during the architectural design phase. That means a utility tree is devised whose leaves contain a summary of the attribute scenarios including the response measure, to which the evaluation of risks are allocated. The depiction is in tree form so that the architect can see at a glance the main category and subcategories ("ISO," 2023; Bass et al., 2022: 210). For most of the subcategories tactics how to deal with the attributes are available. However in some cases the subcategories are proper descriptions.

Scenario ID	Stakeholder Importance	Implementation Difficulty
QA-1	Medium	Low
QA-2	Medium	Low
QA-3	High	Medium
QA-4	Medium	High
QA-5	High	High
QA-6	Medium	Medium
QA-7	High	High
QA-8	Medium	Medium
QA-9	High	Low

Table 4.1: Priority Matrix

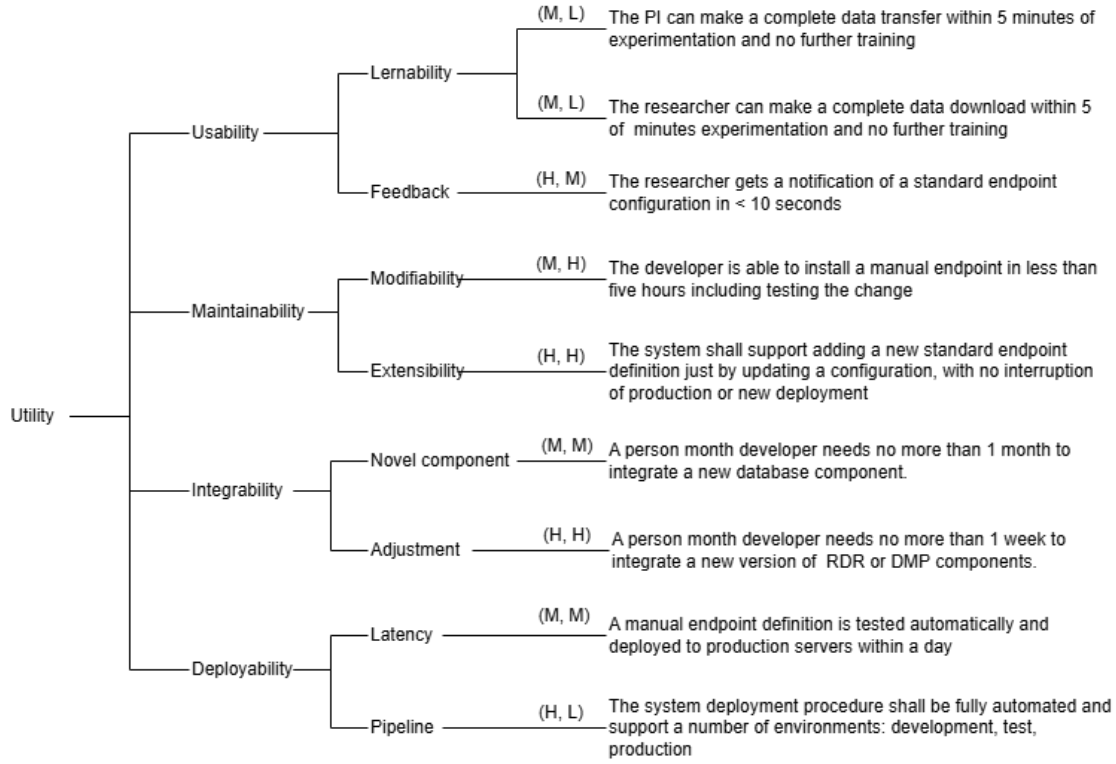


Figure 4.5: Utility Tree

4.4 Primary Functionality

In this section both the functional requirements (table 3.1) as well as the quality attributes (figure 4.5) are considered. From the risk evaluation, attributes revealing low in any of the two dimensions are categorized non-primary because either the technical risk that the feature were problematic to implement or the business risk of how important it is to the stakeholder is low. In either case a failure has the least negative consequences for the final product.

More concretely, all integrability attributes as well as the modifiability attributes are defined as architecturally significant requirements (ASRs) in the sense of Chen

et al. (2013). This only becomes relevant within the design process if the two usability attributes (QA-1 and QA-2) or the one deployability decisions (QA-9) would contradict to integrability and modifiability. In such a case no drawback for the later could be accepted. Still, the non-primary deployability and usability attributes are important quality requirements whose implementation makes a difference of efficient data curation workflows.

The quality attributes can now be related to the functional requirements summarized in table 3.1. QA-3 *feedback* is related to UC-1 *detect faults*. UC-2 *display information* is important for the integrability (QA-6, QA-7) and modifiability attributes (QA-4). So is UC-6 *configure endpoints* but additionally including the latency attribute of QA-8. UC-4 *RDR transfer* is a prerequisite for QA-4 through QA-8 with a focus of QA-5 *extensibility*. The *Log in* functionality of UC-5 relates to all attributes. UC-3 *DMP data* rather aims at the learnability attributes (QA-1 and QA-2).

4.5 Constraints

From the sustainability regulations of RDM centers, in which the architecture would typically be implemented, follows that open source software is prescribed as a matter of principle. Deviations from the rule are possible if no other alternatives are available. Arguments hinting at habitual preferences or the choice of specific measure of quality attributes are hardly accepted. Exceptions are made when security and reliability requirements are crucial. As revealed in the field analysis, the tools applied at all stages of the research data life cycle are plentiful and the majority used in practice is indeed open source so that all libraries or possibly other third party software needed for the final implementation should also have an open source license. As of now, there is no reason to assume otherwise.

A second constraint could not explicitly be found in the needs analysis. Yet, from what can be seen how the stakeholders, researchers and university staff, work throughout all institutions, it is more than plausible to presume the necessity of web browser access. The system must be accessed through a web browser running on *Windows*, *OSX*, and *Linux* since – depending on the institution – installing rich client applications in a local system could turn out to be complicated for services such as fully managed desktops that vary at the different institutions. The software itself must run on a typical infrastructure employed at German universities. In particular, an *Linux*-based operating system and a central relational database server are the minimum requirements. *NoSQL* technologies as a centrally provided service are still not widely in operation when screening the nationwide landscape of university data centers. One could see this as a third constraint.

The network connection to the user workstations may happen to have low bandwidth depending on unpredictable circumstances or longer maintenance task at the university data centers. Though, the reliability of the connection is high. According to consistent reports at data curation conferences, this constraint occurs at the gross of German universities. There are, however, no concrete figures that could be cited and therefore this fourth constraint will be considered if it can be implemented without conflict to other constraints or quality attributes.

Subsystems such as a data repository with an elaborated interface should be in operation at the institution that likes to operate the software component to be developed here. It is a strict constraint since the architecture presupposes a large storage device

for the research data produced at the respective institution with which it is able to exchange meta data and make the transfer of the data.

The most common authentication system in operation at universities as an international average is the *Shibboleth* standard. Although it is not a hard constraint, it bears several advantages in terms of usability for the researchers if Shibboleth is used.

4.6 Architectural Concerns

A development team that should be recruited for the implementation of the architecture often proves to be difficult at public institutions since working conditions, promotional prospects, and payment scales are far from competitive. Especially if the reputation of an institution and the fixed-term contract schemes are seen as a personal investment in a stepping stone for a subsequent academic career. Thus to keep motivation high for several years for a job whose outcome is not contribute to an academic career is a major concern. The programming languages most common in the field for scientific software are *Java* and *Python*. The degree of qualification in these languages differs substantially. If this is the case in the recruited development team it could be challenging to allocate work to its members since the best skilled developer may be overloaded with work while others have little to do. Measuring staff performance and setting incentives fail for regulations in employees legislation at academic institutions.

Besides the specific programming skills, deep knowledge on other technologies such as the shibboleth configuration is necessary to set up the authentication procedure. The concern here is that more time is needed as originally planned for familiarizing with the new technologies. This is particularly relevant when establishing an overall initial system structure as a greenfield development as the present one. The theory in practical software development recommends clearly an advanced team that already has collected experience in brownfield systems or low threshold projects to begin with.

4.7 Conclusions

Design purpose, architectural concerns and constraints follow in principle from the main objective of the thesis and the domain. The quality attributes are different in this respect since they can only be partly derived from the needs and field analysis. The quantifiable properties of the response measure are directly related to the effectiveness of the architecture and should therefore be as sound as possible. To define these measures, the relevant stakeholders have to be put into the same situation as if they used the final product. A workable prototype covering parts of the functionality is as close as one can get at this preliminary stage. The prototype used here is an adhoc version deployed at the university's infrastructure for only one interface that was used for a real life research project. Unstructured qualitative interviews were carried out after a data project was transferred to a test server repository with three scientists.

Admittedly, there is little debate about much more elaborated methods that fulfill criteria such as representativity and objectivity. Yet, it is also clear that such reliable data is expensive to collect. The objective here was to get a realistic starting point from an institution were the software will be used. Depending on which architecture will be developed in the next step, it could be the case that the first implementation

of the architecture could be viewed as just another prototype that is already designed to be extensible if the concrete response measures happen to turn out differently as claimed here.

Such reasoning also applies to the estimation of the risk values. At the end, the risk values are decisive in the prioritization process and thus which quality attributes are given priority on the cost of others. If this decision turns out to be false, other attributes may have been preferred in the architecture. Being aware of possible distortions and have a plan to deal with the consequences when more reliable measures come to light is still the best solution with the available means. To conclude, despite all care to counteract arbitrariness in the selection of quality attributes, the premise on which to build the architecture in the next chapter is to keep it open for change.

4.8 Summary

The most concise summary of the chapter's main outcome is the utility tree shown in figure 4.5. It enlists the quality attributes together with the response measure and the risk values. By means of the tree the ASRs are deductible as the primary functionality: feedback within less than 10 seconds as a usability attribute, modifiability and extensibility as maintainability attributes, integrability in the form of adjusting or integrating new components, as well as deployability within a day for all changes. In combination with the functional requirements, these scenarios will be the most important ingredients for the architecture. Indeed the design purpose must also be clear and it was given in the beginning as a greenfield development in an unsaturated domain and allowing the possibility to continuously extending development from a prototype. The system should integrate research software applications into RDM services by enabling comfortable data transfer by configuration of interfaces. The main architectural concern centers around the organizational issue of having a motivated development team at the cutting edge of technology despite the persistent regulations of academic institutions concerning salaries schemes, data privacy, and other resources. Constraints are evoked by typical restrictions of public institutions. Whereas the commitment to open source software states less of a problem, limitations of the computing infrastructure such as network bandwidth or the offered services of central administered data base technologies are serious constraints.

5
Architecture Design Process

This chapter documents the actual design process, that is, it shows the respective results in three iterations. In the first iteration, the overall system structure is determined. In the second iteration the structure of the primary functionality is defined, in which the quality attribute scenarios are incorporated in the last iteration. At each iteration elements and design concepts are selected, sketched views are given and the design decisions are made explicit.

5.1 Introduction

This chapter comes mainly in the form of tables providing a concise way of understanding the decisions made in the architectural design process. Naturally, there will be some repetitions, i.e. when reviewing the drivers before the first iterations gets started. The table form allows to skip these where needed while keeping the documentation as condensed as possible. Lengthy text descriptions could not give the overview for briefly looking up the one or other element. Yet, it is not possible to completely dispense with text, so that brief explanations are given in a few sentences to follow along the logic in the iterations.

There are three instruments – architectural styles, principles, and tactics – that support the architectural decisions. First, for some of the design decisions, so-called tactics are used to find a way through the complex matter of options the architects has to consider to meet the requirements of a quality attribute. To be clear, how tactics are understood and used here, the following definition is given.

> Tactics provide a top-down way of thinking about design. A tactics categorization begins with a set of design objectives related to the achievement of a quality attribute, and presents the architect with a set of options from which to choose. These options then need to be further instantiated through some combination of patterns, frameworks, and code. (Cervantes and Kazman, 2016: 34)

The way tactics are used here is as a guiding principle accompanied by specific questions. At the end of following along a tree of design principles and answering respective questions, the choice which pattern have to be selected or recombined became tangible. Tactics were used for the modifiability and usability quality attributes.

Second, the categorization of architectural styles and their recommendation depends a lot on the conception of leading scientists in the field. They seems to be less agreement than on the principles and tactics. Yet, common ground is the difference between distributed and layered architectures. To the latter service-based architectures as a very prominent version are usually included here and differentiated from space-based, event-driven, and orchestration-driven architectures. They all are counted among so called service oriented architectures (SOA). Now microkernel, a plugin approach, pipeline architectures, and the modular monolith are all grouped under the term monolith. The classical layered architecture is also part of this group although the flow of control is strictly unidirectional through all layers (usually from the presentation, over business layer to the data layer), which is not necessarily the case for the modular monolith, for instance. Unfortunately, there is some confusion of both terminology and categorization among the authors. To illustrate, Cervantes and Kazman (2016) calls the modular monolith *object model* and would also put architectures based on services into this category dependent on the its deployment while Richards and Ford (2025) follows the schema outlined previously. In this thesis, these terms are used interchangeably with respective references.

None of the confusion could be observed in the application of principles – the third instrument of supporting design decisions. SOLID (Martin, 2018: pp. 57) is the acronym for **S**ingle Responsibility Principle (SRP), **O**pen-Closed Principle (OCP), **L**iskov Substitution Principle (LSP), **I**nterface Segregation Principle (ISP), and **D**ependency Inversion Principle (DIP). All of them are mainly used to receive a clear flow of control, segregate concern, and resolve dependencies were not needed. Really SOLID is meant to be applied at the level of classes and methods. However, the idea behind it can also be applied to the level of components. Actually, one finds the same idea named differently in the software architectural literature. They are called Reuse/Release Equivalence Principle (REP), Common Closure Principle (CCP), and Common Reuse Principle (CRP) whereas the latter two correspond in principle to the SRP and ISP respectively. Since at the level of components (and this is relevant here) dependency between modules is the major factor of producing a *ball of mud* (Richards and Ford, 2025), there are three more principle likely to be used and hence worth mention: Acyclic Dependency Principle (ADP), Stable Dependency Principle (SDP), and Stable Abstractions Principle (SAP). The abbreviations will be used when referring to the respective principle whenever used for a decision or in a subsequent discussion.

5.2 Review Input

The material summarized in table 5.1 below is the most condensed form of what all previous analysis contributed.

Table 5.1: Review of Inputs

Category	Details
Design purpose	This is a greenfield system in a relatively novel domain. Agile processes will be employed to ensure quick feedback by short developmental iterations. Also, a new architectural design needs to be provided, which can be done by substantially modifying existing reference architectures from other domains.
Primary functional requirements	All use-cases in table 3.1 are primary. Non-primary requirements have been sorted out in a separate process (section 3.4.2). – UC-1 Detect faults – UC-2 Display information – UC-3 DMP Data – UC-4 RDR transfer – UC-5 Log in – UC-6 Endpoint configuration
Quality attribute scenarios	All scenarios revealing either a high architecturally implementation risk or a high importance for the users have been selected as listed in table 4.1. – QA-3 Usability: Feedback on configuration – QA-4 Modifiability: manual endpoint modification – QA-5 Extensibility: endpoint added by configuration – QA-6 Integrability: add new component – QA-7 Integrability: adjust RDR or DMP component – QA-8 Deployability: deploy endpoint and test
Constraints	Architectural constraints have been worked out in section 4.5 whereas CON-1 through CON-5 are hard constraints. – CON-1 RDR and DMP are part of infrastructure – CON-2 Open source software – CON-3 Access UI via web browser – CON-4 Linux and relational database – CON-5 Computing center infrastructure – CON-6 Low network bandwidth – CON-7 Shibboleth standard
Architectural Concerns	Described in section 4.6, the architectural concern CRN-1 and CRN-2 are primary. CRN-3 can be addressed with an agile method. – CRN-1 Implementation languages Python or Java – CRN-2 Overall structure in a greenfield system – CRN-3 Leverage technological knowledge among developers

5.3 Iteration 1: Reference Architecture and Overall System Structure

5.3.1 Selecting Drivers

From the review input, it is apparent that all constraints are directly related to the basic architecture of the system. Therefore, all of them will be selected here. The architectural concern that depends on the overall system is CRN-2. Regarding the quality attributes all ASRs impact strongly on the systems architecture. The usability attributes – although they may touch performance attributes, which have not been found to be relevant here – are not dependent on any systems feature. QA-9 is not an ASR and hence will be treated in a subsequent iteration.

- QA-4 Modifiability: manual endpoint modification
- QA-5 Extensibility: endpoint added by configuration
- QA-6 Integrability: add new component
- QA-7 Integrability: adjust RDR or DMP component
- QA-8 Deployability: deploy endpoint and test
- CON-1 RDR and DMP are part of infrastructure
- CON-2 Open source software
- CON-3 Access UI via web browser
- CON-4 Linux and relational database
- CON-5 Computing center infrastructure
- CRN-2 Overall structure in a greenfield system

5.3.2 Element Selection

The entire system is seen as an element in the first iteration (Cervantes and Kazman, 2016: 112, 82). Refinement is done by decomposition following the idea of *Scoping* (Richards and Ford, 2025: 100) in later iterations when the more basic decisions on the overall structure are clear.

5.3.3 Design Concept Selections

There are four basic reference architectures that could be considered as a general design concept: Rich Client Applications (RCA), Rich Internet Applications (RIA), Web Applications, and Mobile Applications (Cervantes and Kazman, 2016: 211). These would fall in the category of monoliths and layered architectures for Richards and Ford (2025). CON-3 excludes explicitly the development of a RCA. But even without this constraint, the experience and advantages of a rich user interface cannot be located at any of the quality attributes. From the primary functionalities – especially UC-5 – it is apparent that no more than three to four tasks per role will have to be selected from a menu so that even a RIA would not deserve the effort of plugin management. The main arguments here are, however, QA-4 on modifying a manual endpoint within five hours and QA-8 on deployability within a day. This is hardly realistic in the type of institution analyzed in section 3.2. These quality attributes can be satisfied with a Web Application, which facilitates updating and deployment. In addition, a web application does not compromise the integrability attributes; on the contrary, in the case at hand, it is likely to promote integration of new or changing components. The usability attributes, although not

primary, rather depend on the clear visual arrangement and presentation of the task to be carried out. As of yet, there are no conflicts to be expected. Mobile applications disqualify here because data transfers are unlikely to be done on handheld devices. Such devices as a main instrument at work are not reported by any stakeholder in the requirements document. Also, at this stage of design, SOAs can be ruled out due to the cost-benefit ratio of distributed architectures.

The next design decision and location refers to the physical structure. The rationale is given by CON-5 as a four-tier deployment pattern (see figure A.1 in the appendix) and more particularly by CON-4 on the decision of the database server and operating system. Thus, web server and application server are run on different tiers, whereas the application server resides in a protected network. The other tiers are the client and the database.

5.3.4 Instantiate Architectural Elements, Allocate Responsibilities, Define Interfaces

The reference architecture depicted in figure A.2 is adjusted to the selected drivers, i.e. it is taken as a reference of elements likely to be used. The rationale to each design decision – how the reference architecture is changed – is given in table 5.2. In figure 5.2 that still reveals a lot of similarities with A.2, the new architecture is shown on the level of modules. The responsibilities for each of the elements (modules) depicted there are enlisted in table 5.3. All this relates to the grey area in figure 5.1 presenting the overall system structure and how the new software is embedded in an RDM and an university infrastructure.

Table 5.2: Instantiation Design Decision

Design Decision and Location	Rationale
Remove local data sources in the web application	There is no need to store data locally. Network connection may be slow but stable.
Redesign Cross-Cutting Elements	These components are either not needed or only needed in one domain object so that a redesign for efficiency reasons seemed plausible.
Remove security layer	This is not a security-relevant application. The degree of security is improved substantially by separating the web server from the application server and by using a well-established authentication process
operation management	These modules are moved to the proxy which serves as the data layer
communication	These modules are moved to UI since they are only needed here.

Continued from previous page

Design Decision and Location	Rationale
Split Business Layer	Elements from the business layer are separated into three different domain objects (DMP, Transfer, Formatter) with logically coherent subsystems containing the business logic, entities, and workflows. Although these elements could as well be organized in layers as the original reference architecture suggests, that follow a clear top-down approach, the use of domain objects ensure encapsulation relevant for QA-4 through QA-7. It enables later changes that are likely to come when additional components are added to the research data infrastructure. And it entails the definition of explicit interfaces which reside in the proxy component. Communication with the client is done via the proxy.

5.3.5 Sketch Views and Record Design Decisions

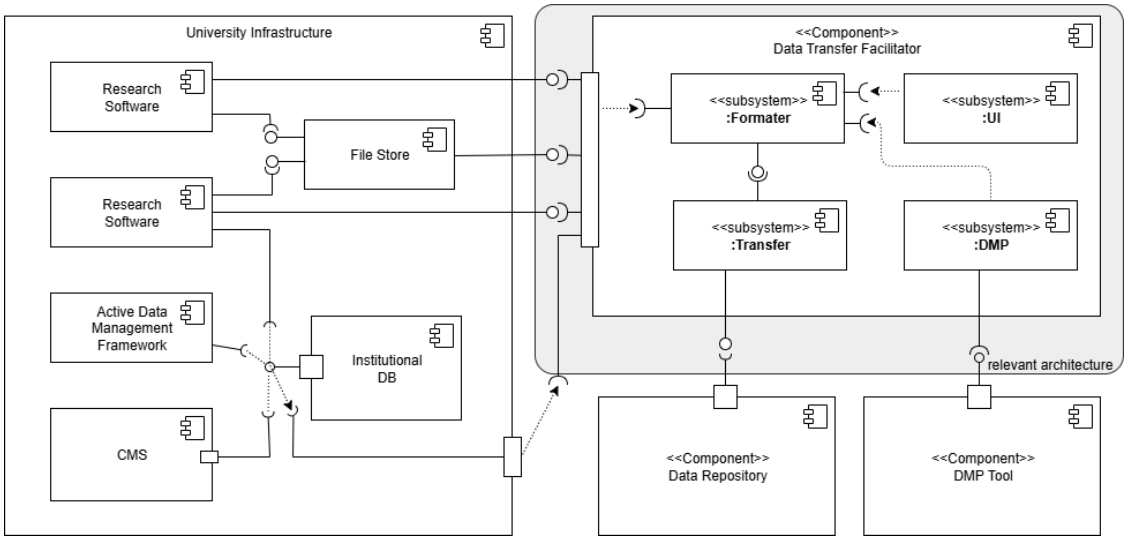


Figure 5.1: Adjusted reference architecture embedded in University and RDM infrastructure

Table 5.3: Design Decision

Element	Responsibility
UI	This domain object contains modules that control user interaction, use case control flow, and services consumed by the clients.
Service Monitor	This module is responsible for displaying logging information and regularly requesting from the operation management whether the endpoint definitions still work.

Continued from previous page

Element	Responsibility
Service Communication	This module manages the communication between the user interface and the results from the business logic. It would also be possible to call it controller since it turned out that it functions like a connecting piece between the logical elements in the business domain.
Service Login	This module is responsible for authentication and rendering role dependent menus.
Transfer	These modules are part of the business logic and handle all actions of the data transfer from the RS applications to the data repository.
Service Access DB	This module contains the specifics how to access the data base, in which project relevant meta data are stored such as the token from the repository or permissions.
Service Access RDR	This module harbors the information and procedure of requesting the repository's interface.
DMP	These modules are part of the business logic and manage the request of getting the correct information from the DMPs.
Service Metadata	This module contains the logic how to process the metadata from a DMP.
Service Access DMP	This module includes the logic to access the DMP interface and to find the correct project DMP given the general information from the login.
Formatter	These modules fine-tune the requests to be forwarded to the repository or the DMP software by mapping them to the prevalent standards if necessary.
Service Format	This module takes care of the formatting of the research data before it will be sent to the repository.
Service Standards	This module maps established data standards to the research data if applicable.
Proxy	These modules correspond to the data layer in similar architectures and manage the correct connections to the research applications, data bases and other persistent storage devices as well as logging information.
Operation Management	This module handles server actions, controls the success and failure of requests.
Service Interface	This module maps the endpoint definitions to be requested from the research applications and the formatted requests of data transactions to the repository or other persistent storage systems.
Service Data Mapper	This module contains the logic to object relational mappings.

Diagram 5.3 shows the adjusted deployment pattern including the authentication server, the repository server, and the DMP server that should be part of the existing infrastructure. The database server and the web server are often also part of the

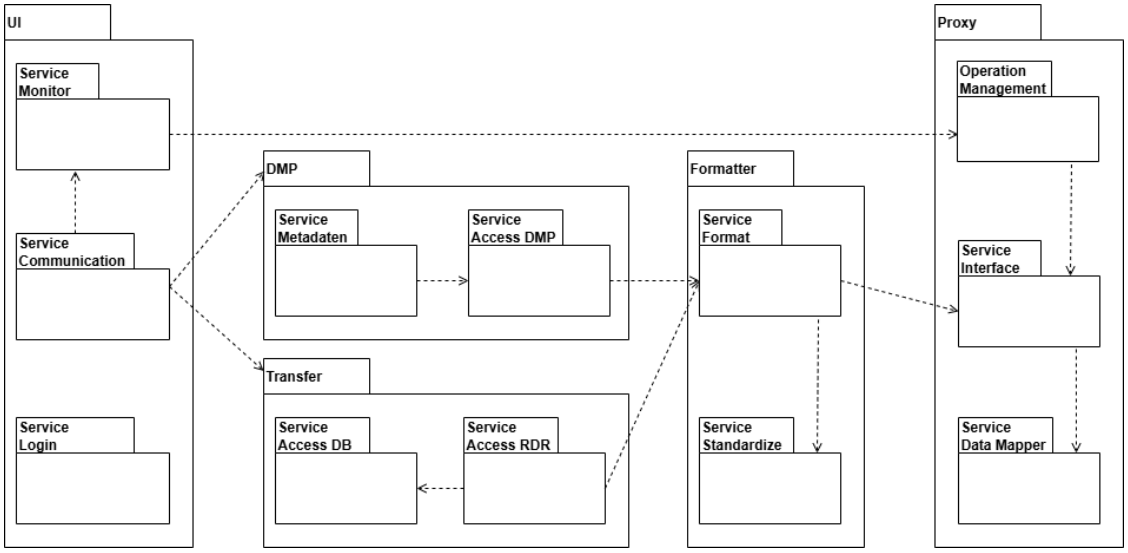


Figure 5.2: Modules obtained from adjusted reference architecture

infrastructure, but have to be configured for a new software project. The application server hosts all server side services and logic as depicted in figure 5.2.

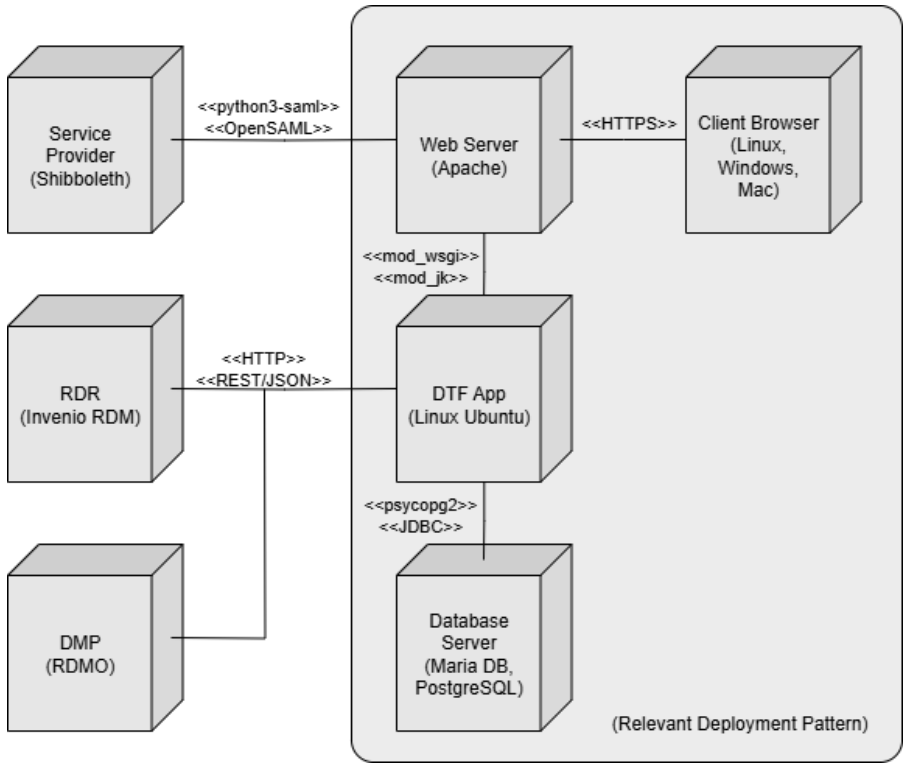


Figure 5.3: Deployment diagram

Table 5.4: Deployment Design Decision

Element	Responsibility
Client/User	The client's Browser that displays the UI and carries out some client side javascript logic.
Web Server	The server delivers the generated web pages.
Application Server	The server hosts the entire program logic.
Database Server	The server that hosts the database to store the RDR tokens and some user meta data.
DMP Server	Existing server that hosts DMPs and metadata of users who have funding proposals.
RDR Server	Existing server that manages a data repository.
SP Server	Existing server that manages users and permissions for a variety of public educational institutions.

5.3.6 Perform Analysis of Current Design, Review Goal and Design Purpose

Table 5.5 enlists the status of fulfillment at the end of the first iteration. Parenthesis indicate that an element was not explicitly addressed in this iteration or it is categorized as none primary.

Table 5.5: Design Analysis Iteration 1

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
CON-1			RDR and DMP are fully considered in the architecture as part of the infrastructure.
	CON-2		Open Source Software are suggested whenever relevant. It is made explicit in the deployment diagram.
CON-3			A respective reference architecture is selected and all modifications made are checked against this constraint.
CON-4			The deployment considers a Linux based system. The database server will offer a relational database.
CON-5			The computing center infrastructure is fully considered in the four tier deployment chain.
(CON-7)			Architecture establishes Shibboleth authentication in modules and deployment pattern.
	CRN-1		Both Python and Java technologies are considered at the relevant level (figure 5.3).
	CRN-2		Architecture so far reveals that no conflicts of quality attributes exist.
		(QA-1)	No relevant decisions made.
		(QA-2)	No relevant decisions made.

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
	QA-3		Architecture suggests that each configuration step done on the server can be logged and made visible to a user.
	QA-4		Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation. An appropriate design pattern below the level of modules will complete this attribute.
QA-5			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.
QA-6			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.
QA-7			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.
QA-8			Deployment diagram establishes efficient roll out as specified by this requirement.
	(QA-9)		Deployment pattern and infrastructural preconditions induce an efficient deployment pipeline and DevOps.

5.4 Iteration 2: Structure Primary Functionality

In the second iteration, it will be necessary to move from the more general overall structure of the system to the specific modules that harbor the implementation of the use cases, that is, units of implementation.

5.4.1 Selecting Drivers

- UC-1 Detect faults
- UC-2 Display information
- UC-3 DMP Data
- UC-4 RDR transfer
- UC-5 Log in
- UC-6 Endpoint configuration
- CON-2 Open source software
- CON-6 Low network bandwidth
- CRN-1 Implementation languages Python or Java
- CRN-2 Overall structure in a greenfield system

5.4.2 Element Selection

The modules depicted in figure 5.2 and described in table 5.3 are the elements to be refined in this iteration. The primary functionality requires the components (figure 5.1) to collaborate with the modules located in the domain objects (figure 5.2).

5.4.3 Design Concept Selections

The reference architectures suggest a layered architectural style. However, this is not implied here. What is aimed at is to learn, which components and modules are likely to be relevant for the green field system. The best trade-off of for a structural pattern to be chosen as an architectural design pattern is the domain object rather than the layered model since more flexibility in the interchange of the modules is to be expected. Also, as concern CRN-2 explicated, it will be a green field system in a relatively unknown domain, which suggests adding additional modules that might need exchange with the data and presentation/service layers in both directions. Indeed, this decision entails interface partitioning, that is, defining explicit interfaces plus a unit functioning as a proxy (Cervantes and Kazman, 2016: 226). Access to the different relational databases can be achieved via the data mapper approach for which open-source libraries (CON-2) in Python and Java (CRN-1) are available. Although constraint CON-6 proposes the implementation of concurrency, the number of projects has to be considered. This is a typical trad-off decision (Richards and Ford, 2025: 30) that the architect needs to make. The risk of failure is thought to be very low when compared with the cost of adding several degrees of complexity by respective concurrent structures into the architecture. The number of research projects permanently using the software system is estimated at a few hundred if at all.

Table 5.6: Design Concept Decisions in second iteration

Design Decision and Location	Rationale and Assumptions
Domain Model Creation	Functional decomposition assumes a domain model showing entities and relationships. There are no real alternatives to creating a domain model and to avoid adhoc architectures with a substantial amount of architectural debt.
Functional requirements are mapped to Domain Objects	The functional requirements UC-1 through UC-6 have to be encapsulated in self-contained building blocks called domain objects so that it is clear which domain object is responsible for which use case.
Domain Objects are decomposed into modules	From the domain model additional domain objects and thus modules will result that have to be integrated into the existing architecture. There are no good alternatives to decomposing the domain model into further modules to support the primary functionality.
Use ORM frameworks for communicating with other systems	Other RSE systems may use very different persistence technologies. Hibernate and SQLAlchemy are considered here. For CRN-1 no other frameworks are taken into account.

5.4.4 Instantiate Architectural Elements, Allocate Responsibilities, Define Interfaces

Table 5.7: Instantiation Design Decision Iteration 2

Design Decision and Location	Rationale
Domain model is initial	The domain model focuses on the use case that is not covered by any other module as it turned out to be the case. It is preliminary because the association to the login is not necessary when looking at the overall architecture, i.e. the information provided with the login is already available in the UI object.
Use cases are mapped to modules	Use case UC-1 through UC-5 could be mapped to the modules that are already present when designing the overall system structure in iteration 1. For UC-6 additional modules are necessary.
Domain objects are decomposed to identify modules with explicit interface	From the domain model and use case mappings, it became clear that additional modules are needed to address explicitly UC-6. The modules *Configurator* and *Configurator Test* are added.
Data Mapper associated with a module in the proxy domain	Object-relational mapping is encapsulated in the service *Data Mapper* module to which the service interface module has access to. The module *Access DB* can also call the *Data Mapper* directly from the domain of business logic. This seems reasonable since it is the only module that needs direct access.

5.4.5 Sketch Views and Record Design Decisions

The first draft of a domain model is given in figure 5.4. This model only comprises the functionalities that are not yet covered in the overall structure from the first iteration. It features how the configuration of endpoints is to work conceptually; it relates to the established modules (*Login*, *Operation Management*, *Monitor*) and defines new modules (*Configurator*, *Configuration Test*) to be integrated into the architecture 5.2.

Figure 5.5 then reveals how the domain model is replicated in the most current design version. This view also show which use cases are related to which modules.

With the new knowledge from the latest design in mind, it is now possible to depict the sequence diagrams for the three most important roles in figure 5.6 for the researcher mainly covering UC-6, figure 5.7 the principal investigator who would transfer the research data as specified in UC-4 and figure 5.8 the IT staff that would receive technical information of the project in UC-3.

In the role of the researcher, a user communicates via the presentation layer, which is handled by the *communication* module. The researcher's main task is to configure an endpoint (UC-6), so the *communication* service will call the Transfer component and within that the *configurator*, in which the access details as well as the endpoint definition must be entered. These are also saved in a database via *Access DB* since they

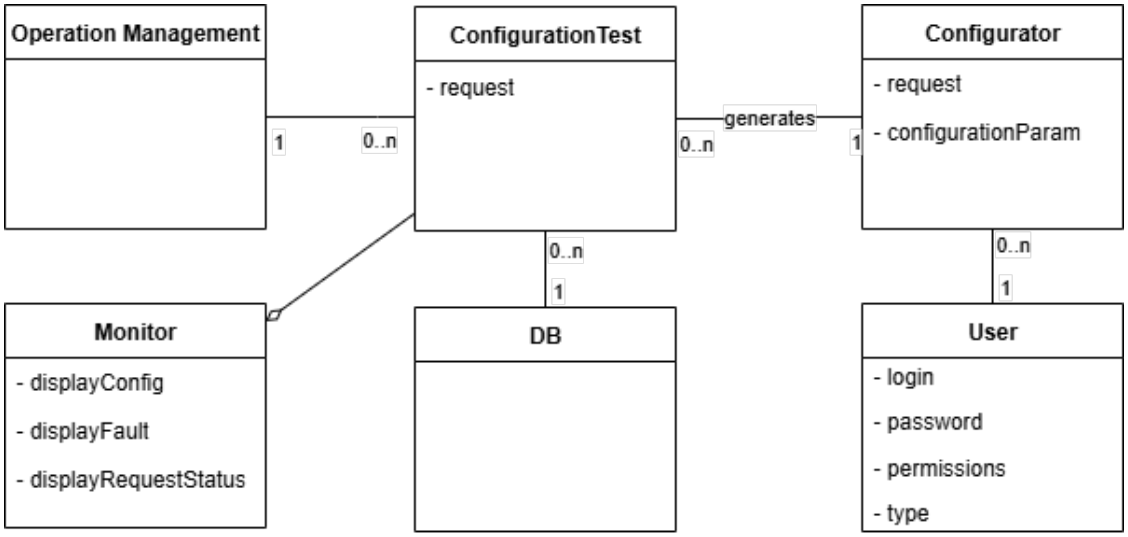


Figure 5.4: Domain Model

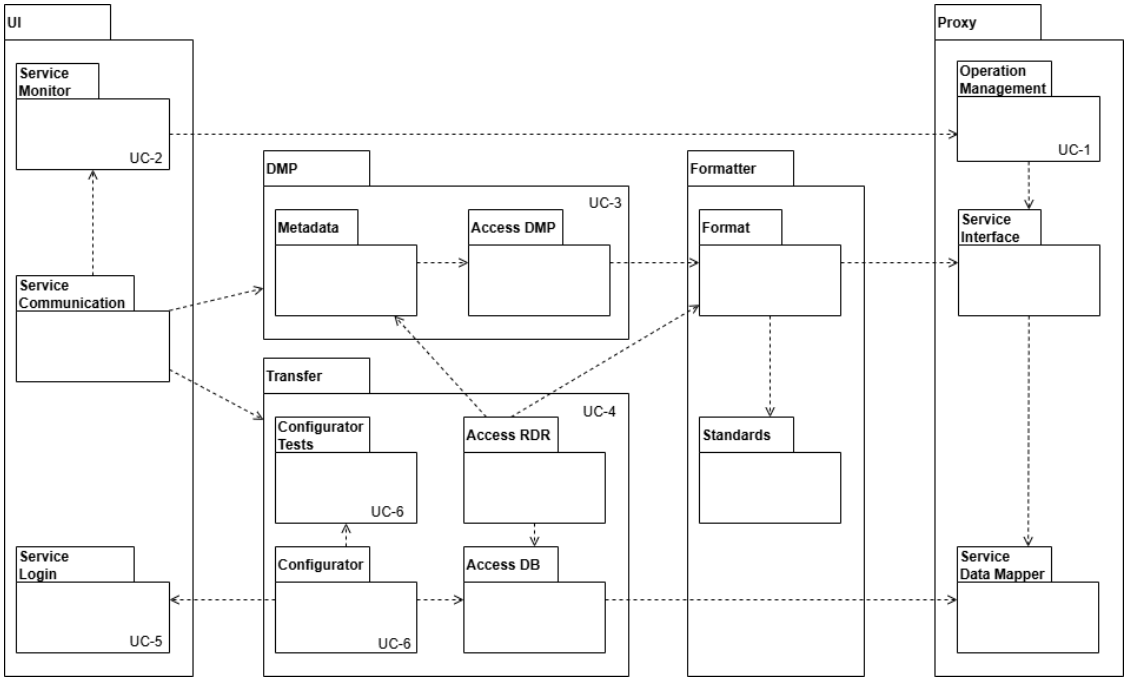


Figure 5.5: Domain Model integrated into Modules associated with use cases

are needed by the monitoring task of the data curator and the PI will have the option to add a repository token. At the same time, *ConfiguratorTest* checks the validity of the definition whose feedback is presented at the UI. For that, the *communication* module initiates the *monitor* service that directly contacts the *OperationsMgmt* module at the data layer. Via the *Interface* the data layer of the research application is reached. Success or failure of the request are reported back to the presentation layer at *Communication*. In case of failure, the Configurator is called and the process has to be redone.

The above process of checking the validity of a endpoint definition is essentially the same for monitoring and configuring for a data steward and will not be repeated. However real data transfer done by the PI follows a different path. The *Communication* service will call the *AccessRDR* that receives access data from the database, but also

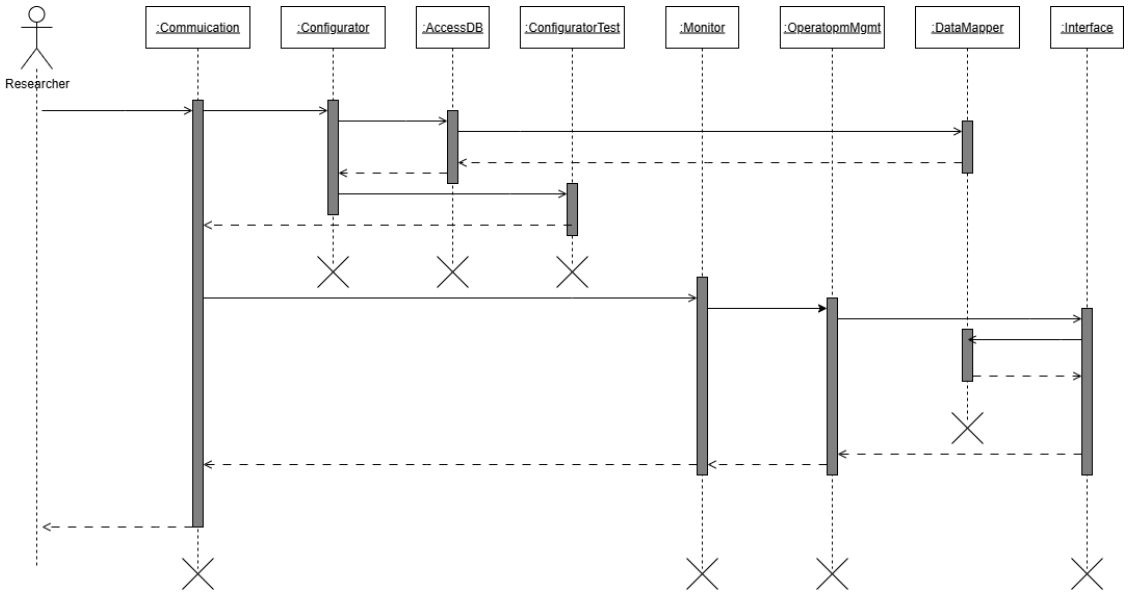


Figure 5.6: Sequence Diagram for UC-6 in the researcher role

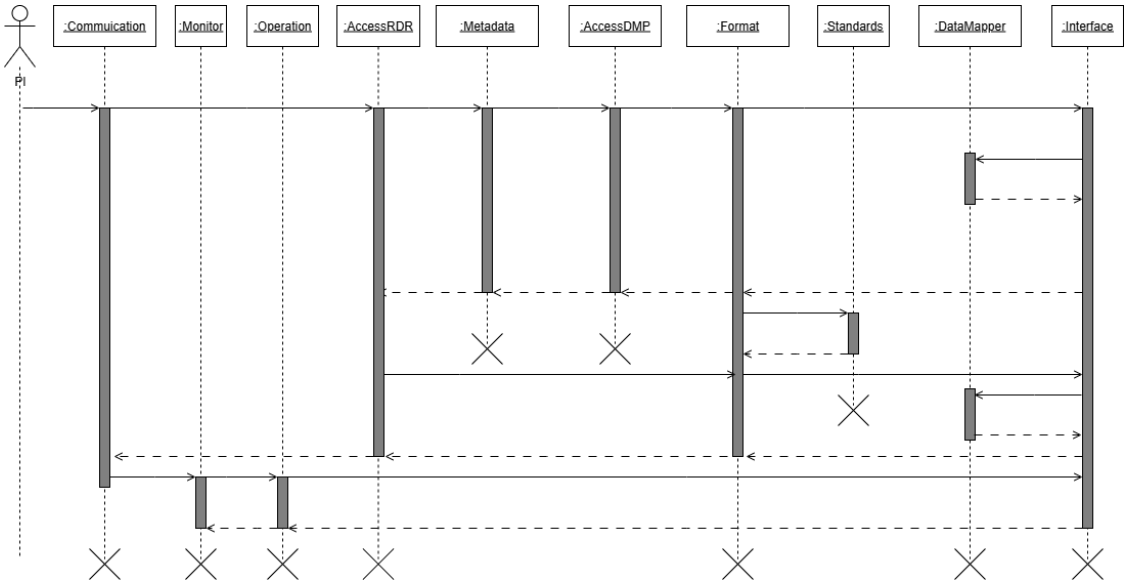


Figure 5.7: Sequence Diagram for UC-4 in the PI role

metadata information from the *DMP* as well as formatting including standardizing if available from the *Formatter* component. The PI may also choose to save the repository access token together with the endpoint definition. Since *Access DB* already has received all access details, it is now possible to use the service *Interface* for data transfer to the selected repository. Monitoring of the data transfer is done via the process described for the researcher via *Monitor - OperationsMgmt* and *Interface*.

The role of the IT staff to have information from the DMP displayed will use a simplified process. Communication will enter the Metadata directly. The module *Format*

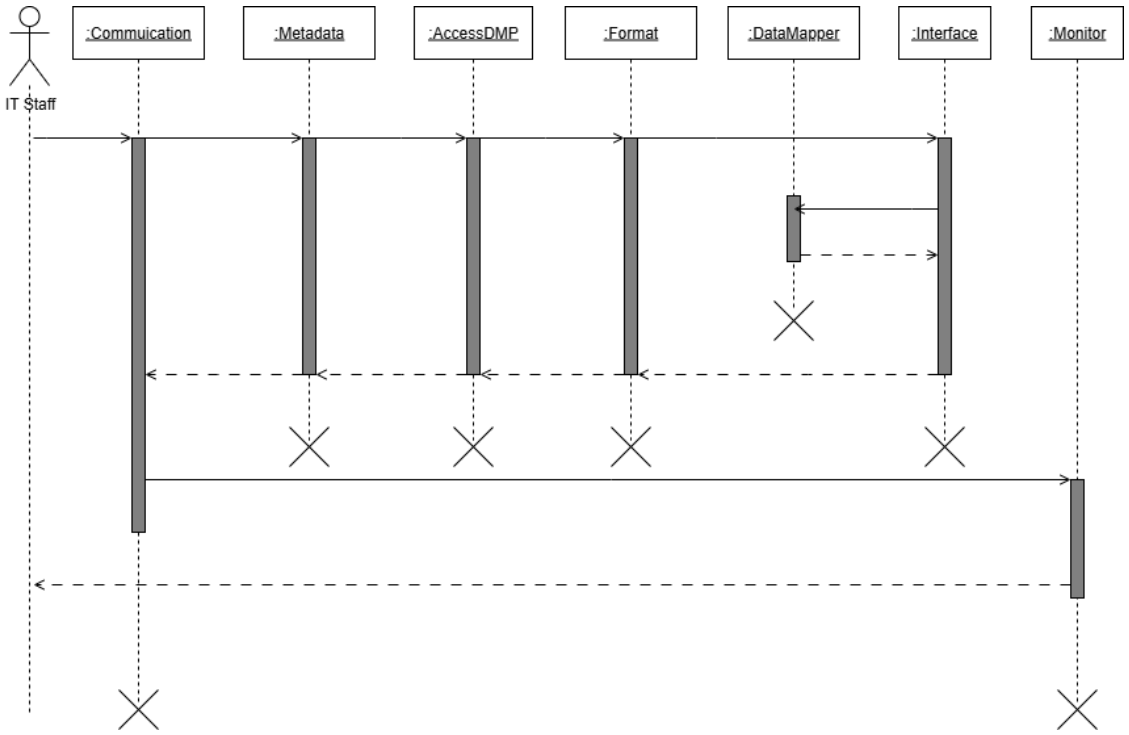


Figure 5.8: Sequence Diagram for UC-3 in the IT staff role

provides only rudimentary formatting[1] before using the *Interface* service to access the questionnaires of the DMP tool.

5.4.6 Perform Analysis of Current Design, Review Goal and Design Purpose

Table 5.8: Design Analysis Iteration 2

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
UC-1			Architecture proposal establishes the modules that will support this functionality (Modules *Service Monitor* and *Operation Management*).
UC-2			Architecture proposal establishes the modules that will support this functionality (Module *Communication*).
UC-3			Architecture proposal establishes the modules that will support this functionality (Modules *Service Metadata* and *Service Access DMP*).

1. Technically formatting could be bypassed because a clear specification of how the data should be presented to the IT staff is not available yet. It would, however, counteract the clarity of the design for not knowing if the data transfer process done by the PI could also take this route.

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
UC-4			Architecture proposal establishes the modules that will support this functionality (Modules *Service Access DB* and *Service Access RDR*).
UC-5			Architecture proposal establishes the modules that will support this functionality (Modules Login).
UC-6			Architecture proposal establishes the modules that will support this functionality (Modules Operation Management).
	CON-2		There are no indicators that any other third party programs will be needed for implementing the suggested modules.
CON-6			Concurrency is unlikely to be needed for the relatively low number of scientific data projects at a university. Even if all projects enter the system at once, bandwidth will be sufficient. As can be seen in figures 5.6 to 5.8 no synchronous processes were sketched to make sure that asynchronous communication is used in a queue acting as a buffer to be work through sequentially, but not in parallel.
	CRN-1		All considered use cases can be realized either with Python or Java libraries.
	CRN-2		Architecture so far reveals that no conflicts of quality attributes exist.

5.5 Iteration 3: Quality Attribute Scenarios

From table 5.8 it reads that all functional requirements have been met. Hence, the third iteration will directly address the quality attributes. As it happened to be a fortunate circumstance, the design decision on the overall structure in iteration 1 have already fulfilled most quality attributes. This has certainly to do with the nature of the quality requirements that are widely requested and thus are reflected in reference architectures. There are more architectural styles specifying a design concept such as the modular monolith, the pipeline, the microkernel, the service-based, the event-driven, the space-based, the orchestration-driven service, or the microservices (Richards and Ford, 2025: pp. 131). In the first iteration the layered style is distinguished from the notion of *domain object*, which may take one of the forms just mentioned.

5.5.1 Selecting Drivers

Since QA-1 and QA-2 have been identified as not to have relevant impacts on the architecture, the focus will be on QA-3 and QA-4. QA-9, as a non-ASR attribute, will be considered as far as it will not lead to unacceptable trade-offs with any other attribute.

- QA-3 Feedback on configuration

- QA-4 Manual endpoint modification
- QA-9 Automated deployment procedure
- CON-2 Open source software
- CRN-1 Implementation languages Python or Java
- CRN-2 Overall structure in a greenfield system

5.5.2 Element Selection

The Transfer component contains the logic, in which configuration is handled. The structure of these modules have to be designed for adjusting code as easy as possible. Extending and modifying the code basis here will also be available for endpoint definitions by configuration, but the foremost goal is to enable adding and changing endpoints effortlessly for the programmer.

5.5.3 Design Concept Selections

The use of the modifiability tactics (Cervantes and Kazman, 2016: 234) give the guidance how best to achieve modifiability as required in QA-4. When extending the source by a new endpoint definition that is not possible to configure out of the box, acknowledged design patterns should be used to minimize the risk of creating a monolithic ball of mud (Richards and Ford, 2025: 133).

5.5.4 Instantiate Architectural Elements, Allocate Responsibilities, Define Interfaces

There are two measures from the tactic applicable: increase cohesion, i.e. semantic cohesion, and reduce coupling, i.e encapsulation, restrict dependencies, abstract common service and the use of intermediaries.

5.5.5 Sketch Views and Record Design Decisions

Although it remains at the specific implementation (and not at the level of architecture), the Abstract Factory design pattern is proposed here as a possible solution to realize the modifiability tactic. There are other design patterns such as *Builder*, *Factory Method*, or *Prototype* that could do the same job – and the programmer is free to choose them – but to be precise a concrete suggestion is necessary. Figure 5.9 also reveals how QA-5 can be implemented. Both QA-4 and QA-5 must draw on the same mechanism. And so will QA-3 as a quality attribute relevant to UC-1 and UC-2 that may come as a messaging service in the operation management module requesting the service interface with the same instantiated methods depicted in figure 5.9.

5.5.6 Perform Analysis of Current Design, Review Goal and Design Purpose

As this turned out to be the last iteration, all use cases, quality attributes, concerns, and constraints are listed for a coherent overview. Besides the primarily addressed QA-3 and QA-4, QA-9 also turned out to be regarded as a solved attribute. No changes to the

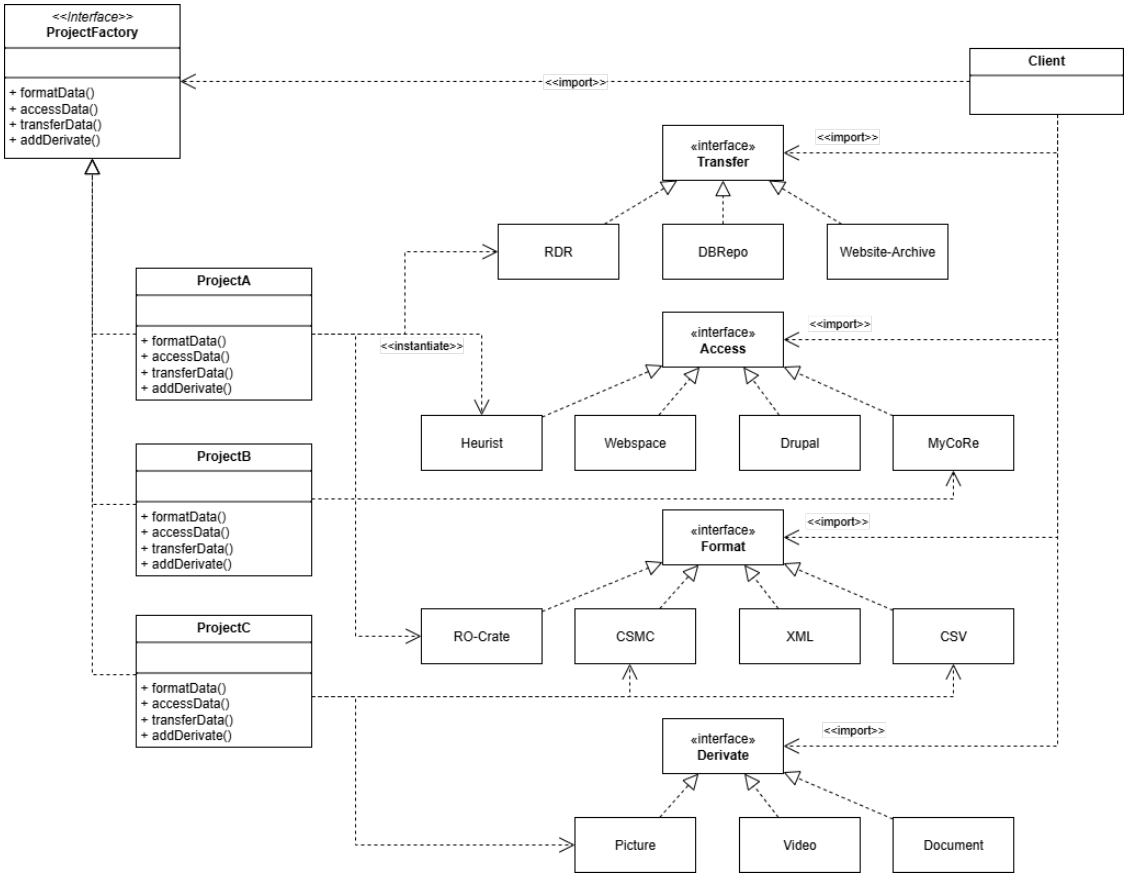


Figure 5.9: Abstract Factory Design Pattern to address QA-4 and QA-5

deployment diagram took place so that an automated pipeline can be set up with the means prevalent at the respective university computing center.

Table 5.9: Design Analysis Iteration 3

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
UC-1			Architecture proposal establishes the modules that will support this functionality (Modules *Service Monitor* and *Operation Management*).
UC-2			Architecture proposal establishes the modules that will support this functionality (Module *Communication*).
UC-3			Architecture proposal establishes the modules that will support this functionality (Modules *Service Metadata* and *Service Access DMP*).
UC-4			Architecture proposal establishes the modules that will support this functionality (Modules *Service Access DB* and *Service Access RDR*).
UC-5			Architecture proposal establishes the modules that will support this functionality (Modules *Login*).

Continued from previous page

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
UC-6			Architecture proposal establishes the modules that will support this functionality (Modules *Operation Management*)
CON-1			RDR and DMP are fully considered in the architecture as part of the infrastructure.
CON-2			Open Source Software are suggested whenever relevant. Most explicitly, it can be seen in the deployment diagram.
CON-3			A respective reference architecture is selected and all modifications made are checked against this constraint.
CON-4			The deployment considers a Linux based system. The database server will offer a relational database.
CON-5			The computing center infrastructure is fully considered in the four tier deployment chain.
CON-6			Concurrency is unlikely to be needed for the relatively low number of scientific data projects at a university. Even if all projects enter the system at once, bandwidth will be sufficient.
CON-7			Architecture establishes Shibboleth authentication in modules and deployment pattern.
CRN-1			Both Python and Java technologies are considered at the relevant level (figure 5.3).
CRN-2			Architecture reveals a feasible solution in the relatively new domain.
		CRN-3	No relevant decisions made.
		(QA-1)	This quality attribute can best be achieved at the level of designing the web front end.
		(QA-2)	This quality attribute can best be achieved at the level of designing the web front end.
QA-3			Extended architecture establishes module *Configurator Test* together existing modules *Communication* and *Monitor* contributing to the usability attribute.
QA-4			Extended architecture establishes module *Configurator* together with the Abstract Factory design pattern that will enable modifiability as proposed in the attribute scenario in figure 4.2.
QA-5			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.
QA-6			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.

Continued from previous page

Completely Addressed	Partially Addressed	Not Addressed	Design Decisions Made During Iteration
QA-7			Original reference architecture based on layers is changed to a Domain Object Model, which eases decoupling and encapsulation.
QA-8			Deployment diagram establishes efficient roll out as specified by this requirement.
QA-9			Deployment pattern and infrastructural preconditions induce an efficient deployment pipeline and DevOps.

5.6 Summary

By using ADD as a method, this chapter dealt with deriving the architectural design from the drivers that, in turn, are substantially based on the requirements. The most concise form of the final result is shown in table 5.9. Three iterations proved to be necessary. The first iteration intended to define the overall structure of the system; the second addressed the functional requirements; and the third iteration took care of the quality attributes. CRN-2 set the general direction in the first iteration giving rise to choosing a reference architecture, which is easy to understand and is reported to work as a general standard. Although the orientation helped to identify the modules that are likely to be relevant for the system at hand, the strictly layered style was abandoned in favor of a domain-based object model that allowed for more flexibility.

Within an approach of component-based thinking, the procedure taken here is described as the *workflow approach* (Richards and Ford, 2025: 113), in which the typical workflow a user may take through the system is mapped to core components (figure 5.1). These workflows were then brought in line with the processes from a reference architecture that best fits the context. Further analysis revealed in the second iteration that this flexibility was needed. Based on a domain model additional modules for handling the configuration logic of potential endpoint definitions were added. Other use cases turned out to be so common that changing the arrangement of and few responsibilities in the modules were sufficient. These refinements are best referred to an *actor approach* in component-based thinking. Workflows and actors are brought together in the sequence diagrams.

Similar to the use cases in the second iteration, the majority of quality attributes could be satisfied with the overall structure designed in the first iteration. The third iteration, then, primarily addressed the manual endpoint configuration (QA-4), which can be achieved with a creational design pattern. By implementing such a pattern, feedback of the success or failure of a manual endpoint definition or by configuration (QA-3) is also solved. Quality attributes addressing the usability attributes (QA-1 and QA-2) will better be managed at the level of front end design. No relevant decision could be made for CRN-3. This point will be taken up in the discussion section (6.3.2).

6 Architecture Evaluation

This chapter addresses the evaluation of the developed architecture. The focus of the evaluation lies in the fit between architecture and quality attributes since these are the significant determiners of architectural design. This feedback is taken as a reference point for a discussion contrasting the evaluation results with other limitations and advantages of the architecture. The guiding principle of the discussion is to look at the trade-offs of design decisions and reflect these in light of the evaluation results.

6.1 Introduction

The evaluation produces results that usually feedback into the architecture and hence these are part of the architectural design process (figure 2.2). For reasons of attributability, this loop is separated here and all possible changes to the architecture is conceived of as the outer loop of the ADD process as depicted in figure 2.4 for brown field systems and so it resembles the process delineated in figure 2.3. Although such evaluation mechanism is not explicitly prescribed in ADD, it is rather advantageous to construe the evaluation by outsiders as a refined architectural design. The main difference here will be that steps two through seven will not be documented as part of a process as done in chapter 5. The steps of LAE reveal, however, that exactly this is done interchangeably with the analysis and team discussions. Thus the outcome, the proposed changes will be presented at its final state only.

The evaluation was carried out as a one day workshop with members of a potential development team and two outside developers with diverse backgrounds in front end, back end and domain knowledge. The steps put forward in 2.4.4 were followed during the workshop. Special emphasis was laid on step three and six since it was clear that all members of the evaluation team knew domain, business goals, and the methodological set up. However, in one respect was deviated from the original evaluation method. Before presenting the architecture in step three, adhoc-solutions were requested from the participants. These were noted down for later reference in step six, in which the discussion could be referred to the initial ideas. This created a much more open atmosphere for other ideas that any previous coining could have prevented. Any

other steps remained as prescribed although for concisely documenting without getting lost in the details of mutual consent, the crucial points of difference were included in the final document.

The advantages and disadvantages of the proposed architecture during the evaluation was captured and it became soon clear that it intersects for some part with a overall assessment of this thesis set aside the arguments of requirements and field analysis. So, to avoid repetition, the critical elements from the evaluation are included once in the discussion section and reflected together with other concerns of requirements elicitation and the RDR environment.

6.2 Evaluation Results

6.2.1 Identified Architectural Approaches

This part of the evaluation process included the presentation of alternative architectures together with their cost relations as estimated in Richards and Ford (2025) and the choice of reference architectures. The preference for a web architecture rather than rich client, rich internet, or mobile applications was straight forward and not questioned. The limited functionality for each role, which can realistically be estimated to less than five menu points left little doubt in the evaluation team to decide otherwise, that is, rich client application also in the version of internet rich clients are characterized by functionalities reflected in elaborated menu bars with often more than ten menu options each. This is the case for internet office applications. More critically, the decision of a modular monolith over a distributed architecture was accepted and considered plausible after a lengthy discussion. Acceptance was granted because a steep increase of additional research applications was not to be expected. The more likely growth rate of research software is seen as moderate given that the prerequisites of depending on the specifics of a university's infrastructure can be projected at the same rate as in the past. More importantly even, the potential to create a SOA from the monolith by laying the foundations in the modules of the architectures was decisive.

The argument of the explicit and obvious advantage of using the DTF for the researcher side was brought forward. The argument was framed as a specific requirement of the stakeholder and could indicate a blind spot in the needs analysis. The other alternative to using the DTF were bypassing the infrastructure and present the data at the end of the project as a *fait accompli* as it is done now. There should be an added value – best in the form of work simplification – to be really used. As it seems now, there is more work to be done on the research side and this counteracts the acceptance of the tool.

Without dispensing the argument, two reasons can be mentioned to counter the it. First, the argument can be identified as a version of Conway's law that roughly states that the communication structures of the institution are reflected in the software. Really, there is no increase of work, but a shift of the point in time when it is done. One could even argue that the opposite is true. The actual work is only made very explicit and thus prevents negotiations of the project's responsibilities by shifting the work to the beginning of the project. In organizations, in which policies do not exist or (intentionally or not) make explicit responsibilities, the better communicative strategies ensure a shift in the workload to the respective other side. In the given case, only the researcher can know how their data can be visualized or which variables have to be included and how

they interplay. As an illustration, a parallel can be drawn to code documentation, which is widely agreed upon to have it integrated in the software writing process. If done at the end of a project after years, it is more time consuming and the workload rises in undue proportion. And it cannot really be outsourced to a third party either.

A second line of countering the argument is to be clear in the organizational processes and disallowing ways around it independent of software usage. Computing centers have service level agreements signed to avoid ad post negotiations of responsibilities. A policy that spells out the data curation responsibilities leaves little room for pushing work in the far future in the hope that it will be organized by itself.

Last, the question of evaluation tooling came up and why the advantages of these are not taken especially because case studies for evaluating the architecture of research software is available (Druskat et al., 2025). The simple justification here was that the tools analyze the source code, derive the modules and its interactions, find cycles and on that basis may suggest improvements. This assumption is not given here, but it is the other way around. The architecture is designed first and only then the source code is programmed in accordance with the architectural specifications and guidelines. Yet, the question is welcomed very much and tools can be applied when the first code implementation is done.

6.2.2 Quality Attribute Utility Tree

Having presented the utility tree containing the concrete figures of the attribute scenarios, initial concerns were raised towards the validity and objectivity of the usability measures. In fact, the time of five minutes that was found in QA-1 was questioned in light of the reviewer's own experience with several similar applications to be rather two minutes. This would also apply to QA-2. With reference to the literature, QA-3 (ten seconds of response time) was evaluated to be realistic together with QA-5 which required no downtime for endpoint definitions by configuration. QA-4, however, was also discussed as arbitrary at first, which could be refuted on the basis of the concrete data collected for the prototype. The later also justified, the deployment attributes, QA-8 and QA-9, whereas the situation with the computing center was emphatically confirmed. This was not the case for QA-6 and QA-7 since QA-6 was only tested for one instance of a database adjustment fairly unknown to the developer. Neither pros nor cons were noted for QA-7 in absence of more concrete usage data, but the underlying assumption of one month were deemed to be in the range of work output given the context at a public institution.

QA-3 initiated a short discussion on an proposedly missing attribute: performance. From a real project, video data about sign language needed to be transferred to the repository, which needed several weeks for manual activation of transfer over night. For such, admittedly rare, projects, scalability will definitely play a role and a distributed architecture is likely to perform better. Load balancing and instant monitoring on separate servers would need to be included in the architecture then. The issue of high-performing data transfer intermingles partly with the collected requirements and its management. While the argument of the performance attribute as such was indeed valid for the given example, the argument blends the added-value for the target group: why is the DTF component attractive to researchers at all if an additional point of dependence, that is, the endpoint definition, has to be maintained? The advantage becomes clear when looking at the entire life span of a project and its resources. For

small-scale data projects with limited resources for technical adjustments, the DTF has the largest savings potential since the usage of the repository interface and likely future changes on it must not be understood. Only changes on the project's data model have to be cared for in the endpoint definition. The same is true for standards and formats. Large-scale projects such as climate or physics data require extra treatment and usually have their own infrastructure at their disposal. Well trained IT personnel is also available on a permanent basis, so that the interface of the general storage infrastructure can be transferred directly and also maintained in the long run. Thus, the question addressed in the evaluation was then an respective infrastructure would pay-off for including the very exceptional cases. However, the evaluation pointed to a more thorough and precise definition of the target group and the kind of projects that are supported by the architecture.

6.2.3 Prioritized Scenarios

The prioritization of quality attributes is based on the implementation experience and the importance of the attribute for the user. The simplicity of the risk values was assessed as positive. If a performance attribute would be confirmed to be a real requirement, the prioritization were likely to be marked as *high*|*high* in the risk matrix and so prioritization had to be changed. As a consequence, the performance attribute could challenge the maintainability attribute and the design decision could shift in favor of a distributed architecture. Otherwise all scenarios could stay as they were.

6.2.4 Analysis of Identified Architectural Approaches

A consistently positive feedback from all attendees of the LAE was given for the decisions made in the first iteration, that is, first, to construct the architecture from references that reveal similar requirements and that are recognized by the software engineering community and, second, make adjustments in the layers and rearrange the communication between modules in the layers by decomposition. Although the second step changed the nature of a layered approach, in which information be carried only to the very next layer, the new overall system structure was considered clear. It was raised that it could be redrawn so that the domain objects could be more freely arranged on the cost of the familiar structure of presentation, business, and data layers. The advantage were to better see superfluous associations and possibly find better ones. Yet, that the given architecture is prepared for unforeseen changes is fully accepted since the identified modules can easily be grouped as independent services of a distributed architecture due to the few associations between them. The extensions made in the second iteration based on a domain model that would only reflect the core functional requirement of the configurator was approved due to the very obvious interrelationship of the module and specified requirement. So the mapping of the use cases to packages and modules was acknowledged by the evaluators.

Having presented the result of the third iteration which incorporated the quality attributes that had not been resolved previously, the question came up how exactly ensured the architecture 10 seconds of response time in QA-3? Since this figure should be kept separate from the transfer time, with which it may had been confused, the *Operation Management* module in the data layer would provide respective measures of the state of any transaction made in *Interface* service. The *Communication Service*

displays the feedback from the *Monitor* service requesting *Operations* every 10 seconds. This mechanism was designed intentionally and it is the reason why the strict top-down architecture of layered systems was resolved. The transfer time depends on the complexity of the endpoint definition as well as the size of data to be transferred or downloaded so that a meaningful limit value cannot be set. The progress of a transaction, however, could be monitored. This feature will be taken up.

This discussion pointed out two further issues. First, the use of asynchronous associations was criticized as not necessary. The argument here went that asynchronous information exchange is not needed as far as the sequence diagrams showed. The information can be sent synchronously without a difference. Asynchrony between the services would require a much elaborated operation management as implied in the given architecture whose expense would probably not be justified. After consulting the literature, the point was taken and agreed upon. Second, as a left over from the layered reference architecture, the components of the business logic reveal arrows whose direction went into the data layer of the interface service. This clearly violated the DIP because changes in the services *Data Mapper* or *Interface* necessitated changes in the business logic – the part of a software that should not be changed at all. Despite the fact that a design pattern were an implementation detail not belonging to the architectural work, a solution[1] was worked out specifying the abstract factory pattern for the business logic that guarantees DIP via explicit interfaces. Improvements can be done by further separating the selected formats (*XML*,*CSV*) from the *RO-crate* and *CSMC* standards. The advantage of this solution can also be justified on the grounds of an additional functional use case, i.e. if the DTF is to be used as an interface to reuse the data sources for further analysis once uploaded to the repository. Put differently, the direction of data transfer could be reversed.

The discussion on the abstract factory design pattern initiated yet another point. The *Configurator* that resorts to the abstract factory should only be available to the role of researcher and data curator as it also became clear from the sequence diagrams in figures 5.6 through 5.8. The ISP implied that a different user interface should be set up for each of the roles to be prepared for a change concerning only one role, for which it was not clear how it interfered with the logic of the other roles. This detail was not yet made explicit in the descriptions of the architecture. The comment was much appreciated and assured that separate interfaces will be implemented. At the level of architecture it was uncommon to give these specifics.

6.3 Discussion

The evaluation revealed the nature of trading-off in software architecting from a hands-on point of view nowhere else during the architectural work as tangible as here. A collection of pros and cons that are immediately apparent are not only backed up by third party perceptions, but also some previously unconsidered limitations could be identified. Even more notably, it became evident that the principles as listed in 5.1 may not all hold and so it follows that the art of good architecture is to compromise between the advantages and the disadvantages. The counterbalances cannot be shifted arbitrarily to either side. The best that can be done is to name the balance and reflect upon it until technological innovations produce better solutions.

1. It is now already included in the third iteration as figure 5.9.

6.3.1 Advantages of the Proposed Solution

From the beginning of working on the architecture, the statement that change is inherent in software has been accepted as a guiding theme. Looking at the final evaluation document, the degree of modularity is generally perceived as high. The use cases are clearly attributable to single modules. The object model can be extended and the resulting components can be allotted to the architecture, whose overall structure is consistently visible in all diagrams. Thus, one may acknowledge the system is open for change and this is an advantage that architectures constructed from scratch have. Those do not have to harbor legacy issues. As long as the principles (5.1) are obeyed, the system will stay open to change. Change is built into the structure. It is the first advantage.

Along with this comes a second advantage. The internal structure of the system does not necessarily remind on a modular monolith per se. It is a monolith because a single executable is available. But it would easily possible to add a executable to each of the components and have their modules act independently as in any of the distributed architectural styles. The modules themselves are designed as services already. Yet they still act within a single runnable. Refactoring the prevalent architecture to a distributed style should not take many resources and time. One could even argue that it is loosely spelled out for future versions. Such plan could be put into practice if the users accept the system and openly indicate that the system became an indispensable tool in their working routines. In case usage behavior remains low, the much more economic monolith is further maintained.

While strictly following the single responsibility principle or the common closure principle respectively, separated concerns can be ensured and this produces the aforementioned advantages. A third advantage results from the dependency inversion principle at the level of classes and modules and acyclic dependency principle and stable dependency principle at the level of components for a largely independent, highly decoupled business logic. These rules, handling how the data is processed and transferred, stay as is if the presentation or interface specific code changes. This is a major contribution to the maintainability of the application. Especially the presentation layer is likely to change several times. Keeping the business logic independent prevents the developer to touch the core logic and further minimizes errors in the central concern of the application.

The fourth advantage is that the architecture is built on identified quality attributes. In fact, quantifiable attributes form the backbone of the architecture. Although modifiability of these is not easily feasible, other quality attributes that may become relevant in the future could be extended. To be fair, this process depends on the attribute. Security, on the one hand for example, would require massive reorganization from scratch, which could justify a complete rework. On the other hand, scalability should still be implementable with less effort when converting to a distributed style. The distributed architectures, however, are more costly in operating the involved resources at the physical layers and in general maintenance, so that the trade-off of such decision has to be counterbalanced to the relevance of scalability. A similar case could be made for performance, which is often cited as an important attribute as the major impact and success factor on the acceptance of a web application. Despite the fuzziness of which attribute may become relevant in subsequent versions, the circumstance that the architecture considers quality attributes in the first place and is designed on this basis, is a factor that should not be underestimated. Especially when redesigning brown

field systems that treat quality attributes as a side effect. The architecture at hand is documented in such a way that a future redesign can be paralleled with a different set of requirements. The architectural concern (CRN-2) expressing the insecurity of applying a new overall structure in an unknown field is largely diminished for the experience with the existing system.

A last advantage has already been implicitly alluded to in the previous paragraph. It is the low cost of monolithic architectures. Indeed, cost are relative. They are compared to the most probable alternatives and these are plug-in and distributed styles. Part of the cost function (but this can also be separated) is the understandability of the software system. Monoliths per se are reported to be easy to understand. The many intricacies of understanding asynchronous processes in distributed system can be dispensed with just as making sense of plug-in dependencies. The system presented here is no exception especially since the degree of modularity is high and concerns are clearly attributable. This advantage is relevant if developing teams are not able to keeping pace with technological advancements.

6.3.2 Limitations

No satisfactory result is achieved on the third architectural concern (CRN-3) about leveraging technological knowledge among the developing team. The literature reveals that because of modular monoliths were considered domain-partitioned architectures, teams should also be aligned by domain area (Richards and Ford, 2025: 174). Thus if a high level of modularity and separation of concerns is given in the monolith, an *enabling team* approach (Skelton and Pais, 2024) would foster learning, collaboration, and proactive behavior. This idea is directly taken from the theory and part of the architecture was designed with this idea in mind – it is as modular as it can be, concerns can be subsumed to each module, and all of this is strictly derived from the domain model – but it remains totally open to the architect how this actually has an impact on the still restricted technological knowledge of the team. At the end, the development team must be instructed to incorporate learning of technological knowledge in their daily routines. Parts of this knowledge must also be tested in real-life projects and error tolerance is to be widened. This, however, cannot interfere with critical services, that is, data loss for example is not an option. How to exactly spell this process out is totally unclear and it certainly is a limitation that the architecture is not able to capture as implied in the literature just cited.

A second limitation is the restricted domain of applicability. Although the field of RDM of public universities is widely homogenous nationally as well as internationally and has very similar infrastructure requirements, the domain as such is restricted to a couple hundred individual players. All prerequisites of a typical RDM infrastructure, such as data repositories and other centrally administered data services have to be in operation to implement the DTF system. These tight limits narrow a general usage at other organizations that have similar goals in the managing a wealth on distributed data applications. Part of the limits are the constraints and concerns the system is designed with. As an illustration, the early choice of the programming language (Java or Python) is evident from the surveys at public institutions and must therefore be taken into account, but it lead to neglecting other technologies. At least, technologies that cannot be realized with libraries in Java or Python were not looked into any further.

A third limitation is a methodological issue in requirements elicitation. On the one hand, the field analysis represents by and large the landscape of RDM infrastructures at least all common characteristics prevalent at a common German university as it is reported in several surveys. On the other hand, the needs of the identified stakeholders could only be collected from very few data projects at one institution. These requirements entered as use cases and concrete measures in quality attributes the architectural design. Despite the fact that this procedure is reported as a common compromise in many software engineering practices, it leaves behind a distortion that cannot be dismissed. Of course, it is possible that a representative study based on a random sample of universities and its stakeholders surveys the same needs, but it is more likely that the responds are more diverse. It remains a blind spot. A representative study, however, would not justify its expenditures if compared to the cost of developing the architecture.

Related to this issue, but only indirectly relevant for the present and future development of the architecture is the fourth criticism. As it turned out in the prototyping phase, not all stakeholders can be contacted reliably. Some stakeholders involved in the pilot study have already left the institution, others are still available and work in other projects. It would give valuable insights if the implemented system could be tested on the same stakeholders before publishing the first code release. Such interchangeable development would foster the effective implementation of QA-1 and QA-2 on the usability of the user interface and it could provide valuable hints for acceptance. In future releases, contact information of stakeholders and their agreement to give feedback on versions in progress would have to be made to receive similar positive effects.

6.4 Summary

This chapter covers two related topics: the evaluation and the discussion of the proposed architecture. The evaluation was designed as the lightweight architecture evaluation (LAE) as proposed by Cervantes and Kazman (2016). The fruitful discussion at the last phase of the method set the incentive to elaborate on the points brought forward there and therefore have it all put together in an extra chapter. The evaluation also includes the improved architecture.

7 Conclusion

The task of the final chapter is to wrap up the results of thesis, reconsidering in how far the objectives have been met, and view the result in the light of prevalent research data infrastructures. A bird eye perspective suggests that the proposed architecture can be seen as yet another part of a larger family of architectures subsumed under the label of micro-services. The second part of the wrap-up gives an outlook of future work. It includes the idea of the concrete implementation of the architecture and possible extensions that may be relevant in the future, if research applications were considered to be fully maintained or other challenges such as new requirements for security or scalability had to be faced.

7.1 Results in the Light of RDM Services and RSE Practice

Having worked through the mighty field of software architectures and applying ADD to a real world problem, it is now time to take a step backward and review both result and process. Changing the altitude of observation, the overall RDR software infrastructure including the DTF appears to have a different architectural style than the specified solution of a domain partitioned and modularized monolith. From such a distant perspective the involved research applications appear to be part of a micro-service architectural style characterized specifically in this case by isolating part of the data layer as an extra service, that is, fulfilled by the DTF. This extra data service is only relevant for research data that changes little or not at all. The other part of the data layer, that is, data that undergoes change, that is, incomplete research data, will use a different service with another isolated database, that is, the project internal data space with an interface to the data layer service (also part of the DTF). Thus, the original service that treats all data the same is split into two services that distinguish between the properties of the data. To stay in the picture of a bird's eye view, the DTF also acts as the API layer guaranteeing the full decoupling principle of micro-service architectures.

The thesis makes no statements about the sustainability of the single research applications, how to maintain and keep them alive. These are fully independent units that are loosely coupled via the interface. Conceptualizations and concrete strategies

on this topic are still subject to the ongoing research and debate (Felderer et al., 2025) (section 2.3.2). The approach taken here is to keep research software as units of their own right that have the possibility to make contracts to the interface, which makes them part of a kind of micro-service whose overall functionality does not depend on the single research software. Rather, the common denominator is the produced research data and possibly on the category (Hasselbring et al., 2025) to which the standards of sustainability are applied to.

From this, a prerequisite of RSE practice becomes clear. To have the proposed solution work, the research software engineers are not entirely free in their choice of the technical infrastructure. In fact, all projects have to be part of the institutions' infrastructure. As a minimal requirement, an accessible data base server or other from the outside accessible storage devices have to be used, so that a minimal contract in the distributed system can be signed. Best, the research application uses a centrally administered data base server and, for storing other relevant data, clouds and webspaces, to which read access, based on different user roles, can be granted.

Revisiting the three research questions put forward in the beginning of the thesis will allow for a critical assessment of the achievements of the work done. The first research question aimed at the domain knowledge, i.e., most importantly the requirements that were derived from the field analysis. The field analysis proved to be more challenging in that an average base line of properties and needs, which are relevant to all institutions, was attempted to be found. It goes without saying that such an endeavor cannot incorporate all peculiarities deemed to be important at a single institution, but turn out to be outliers when considering the entire field. And even such statement is hard to prove since a representative study including the definition of the *field* on this is still missing and the decision on what to include and exclude is admittedly biased. However, considering the alternative of taking into account only one or two institutions is likely to produce even more biased results. The chance that exactly the institution at hand is representative to all others is certainly lower. Scanning through the landscape of RSE applications, there is indeed the tendency of developing a specific software for the home institution and releasing it as open source. Other institutions with similar needs may fork and adjust the software. So, it spreads in a bottom-up fashion. One can also observe on the wealth of such software that developing teams tend to inflate the original software without caring too much for the suggested design patterns. Others, again, rather create their own solutions without taking full advantage of the existing systems. Here the assumption is that the prioritization of quality attributes is inaccurate. Considering these arguments, the approach taken in the thesis at hand can be the opposing alternative. Yet, it is not equivalent to a top-down approach; it still remains bottom-up. The difference is that only the very core of requirements at an average university without institutional idiosyncrasies are included in the architecture. Because the research data life cycle is recognized by the majority of research data management centers as a best practice and the architecture is built on this domain, a common basis of requirements should be better applicable than adjusting from a specific instance, which may hide isolated implementations. In light of all this, the research question is answered to the degree of a limited set of requirements that are plausibly traceable to the field analysis. The field analysis itself is based on the material that is available as of today, and so also this part of the question is achieved.

The second research question addressed the definition and prioritization of quality attributes. Following the style of the established standard, scenarios were worked out

that reveal quantifiable measures. While the quality attributes themselves were derived from both the functionalities and the field analysis, its quantization could only be done and tested for selected individual projects via prototyping. The pitfall here is obvious and it hardly deserves an elaborated discussion for missing the convincing counter arguments. There is a clear disadvantage because none of the plentiful surveys would go to the level of detailed questioning that would be necessary to have reliable figures for maintainability, deployability, or usability. And that is very understandable. From another perspective, one could say that the relevant quality attributes could be defined at all is a success. Factoring in the FAIR principles in section 2.2.1, its reference to RSE (2.3.3) as well as considering the properties of research data (2.2.3) and research software (2.3.1) is necessary, but not sufficient. It could even be misleading if one only considers what follows from the theoretical background. To illustrate, the concepts of interoperability, accessibility, or reusability do not correspond to the quality attributes of the same name. These must be translated in the same way as the institutional requirements (2.2.4) on compliance and copyright. To find out what is really needed in terms of a quality attribute, represents the actual merit of answering this research question. To realize that it is rather modifiability than performance and scalability together makes the difference. So what can be observed here is again a compromise between imperfect knowledge and feasibility. Taken it all together, quantifiable quality attributes could be defined. Usage behavior of an implemented DTF will show at last if the quantization of the attributes were suitable for this purpose. Until then, the research question is considered answered.

The goal of the third research question was to produce a software architecture. The method to be used was attribute-driven design since recent research has hinted consistently in this direction. Since architectures are trade-off decisions, emphasis was put on minimizing conflict and always with regard to the prioritization of the quality attributes. All in all, following the steps prescribed in ADD, the iterative processing gave the guidance to achieve steady progress without being sidetracked by secondary decisions. This is not to state that the outcome to arrive at a good architecture was clear. The respect of the huge body of technical knowledge an architect must have is still overwhelming and a rest of insecurity of having made suboptimal decisions remains. And this is the case, even though all decisions are rationalized and reasoned with concrete justifications. The deep understanding of the domain knowledge continues to be the most important impact factor and the most exact predictor for the architectures overall quality. But the overt state of uncertainty as one has to recognize in the previous research questions – that are open to see though not to change – is not encountered here. The existence of one appropriate architecture does of course not exclude the existence of several other appropriate architectures, in which trade-off decisions have been made differently. To give a final qualitative statement, also the third research question is fully answered.

7.2 Future Work

The first next step is to put the architecture at hand into practice. To be realistic, the full implementation entails financing the enterprise, which is typically done in the public sector with the application for qualified funds. Provided that funding is successfully obtained, further organization will be invested in recruiting and managing appropriate units to an existing development team that could then further work out how to achieve

the implementation by agility. Some of the issues emerging with operationalization have been addressed in the architectural constraints as well as concerns.

Two quality attributes related to the front end's usability are yet to be resolved. This can be done in the process of designing menu structures and clear arrangement of the web interface. Feedback in the form of lightweight usability test from the stakeholder groups of researcher and PI will have to be carried out. This procedure should also involve asking for the duration of transferring a test data set.

The architectural concern of knowledge and learning in the development team remains crucial throughout implementation time and beyond. The long-term goal is to create a sense of self-awareness for constantly keeping up to date with new technologies and make clear that time must explicitly be set aside for this. Agile project management, in which teams are asked to organize and plan learning sessions on their own responsibility as part of their daily routines, is an effective means to create the incentive for such behavior. Financial means will have to be provided for regular workshops on topics that the agile team agrees upon.

Once a productive version of the DTF is available to the research community, the quality of the stakeholder analysis and requirement elicitation will be subject to a litmus test: the success of the project can be measured against how many researchers of the institution will actively use it. And this will crucially depend on how exact the needs are translated into the implemented requirements. Assuming that – as it is done here – there is a steadily growing number of users, the likelihood of new requirements is high to occur in two ways. First from the users, other needs can evolve and, second, the application itself will change for the emergence of new technologies, other traffic and user behavior, security issues, and many more. So other quality attributes, such as performance and security, become relatively more important than at the start of the software. Change will be inevitable. When the application keeps on growing over time, other principles of software design will become more relevant and changes will have to be made (Martin, 2018: 109). Certainly, future work will go in this direction. And the architecture as it is now, is prepared to change as good as it can be under the conditions of today.

> If the system's architectural direction is still unclear, it's often more effective to begin with a modular monolith and later move to a more complicated and expensive distributed architecture style, such as a service based [...] or microservices [...], than to jump straight into the distributed architecture. (Richards and Ford, 2025: 176)

Provided that change is an inherent characteristic of all software architectures, the quotation delineates a possible track of change. Steady change means keep architectural decisions delayed as long as sensibly possible (Martin, 2018: 140). The modules of the modular monolith as of now have already been designed as services largely independent although kept within a single component. The exclusion of these services should be feasible with minimal effort. New service modules could be added to this structure. It is easily conceivable that e.g. load-balancing could play an important role in the future. This could particularly be the case if huge amounts of research data of several projects are expected to be transferred in parallel. Other future scenarios predict research applications as part of an established micro-service with shared software libraries that are centrally maintained along with standards that the research application would have to fulfill on their side. Thus, none of the research questions recapitulated above is supposed to be perceived as finally assessed. In fact, none of them is closed.

From only roughly observing the history of software systems and their genesis, changes of the domain, the entire field of research software and research data infrastructures, are as probable as those in the functional requirements of a rapidly changing society. The two are mutually dependent. Once the field, the domain, the needs, the expectations, and hence the functionalities undergo changes, the resulting quality attributes and the architecture will also undergo changes, with the former following the latter. And yet, there is little more to add as future work will continue where this project has started.

Appendices

A

Additional Material

APPENDIX

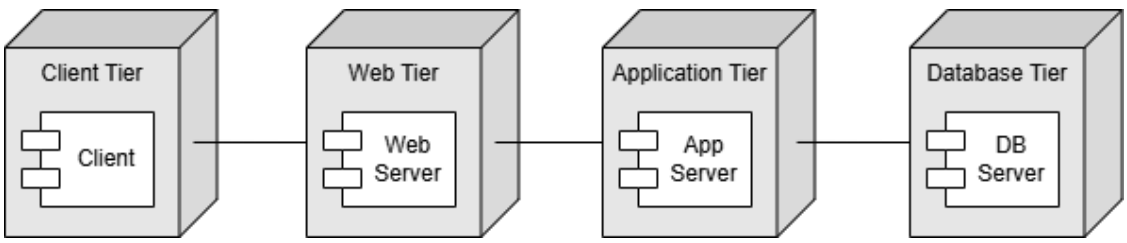


Figure A.1: Four tier deployment pattern (Source: Cervantes and Kazman, 2016: 223)

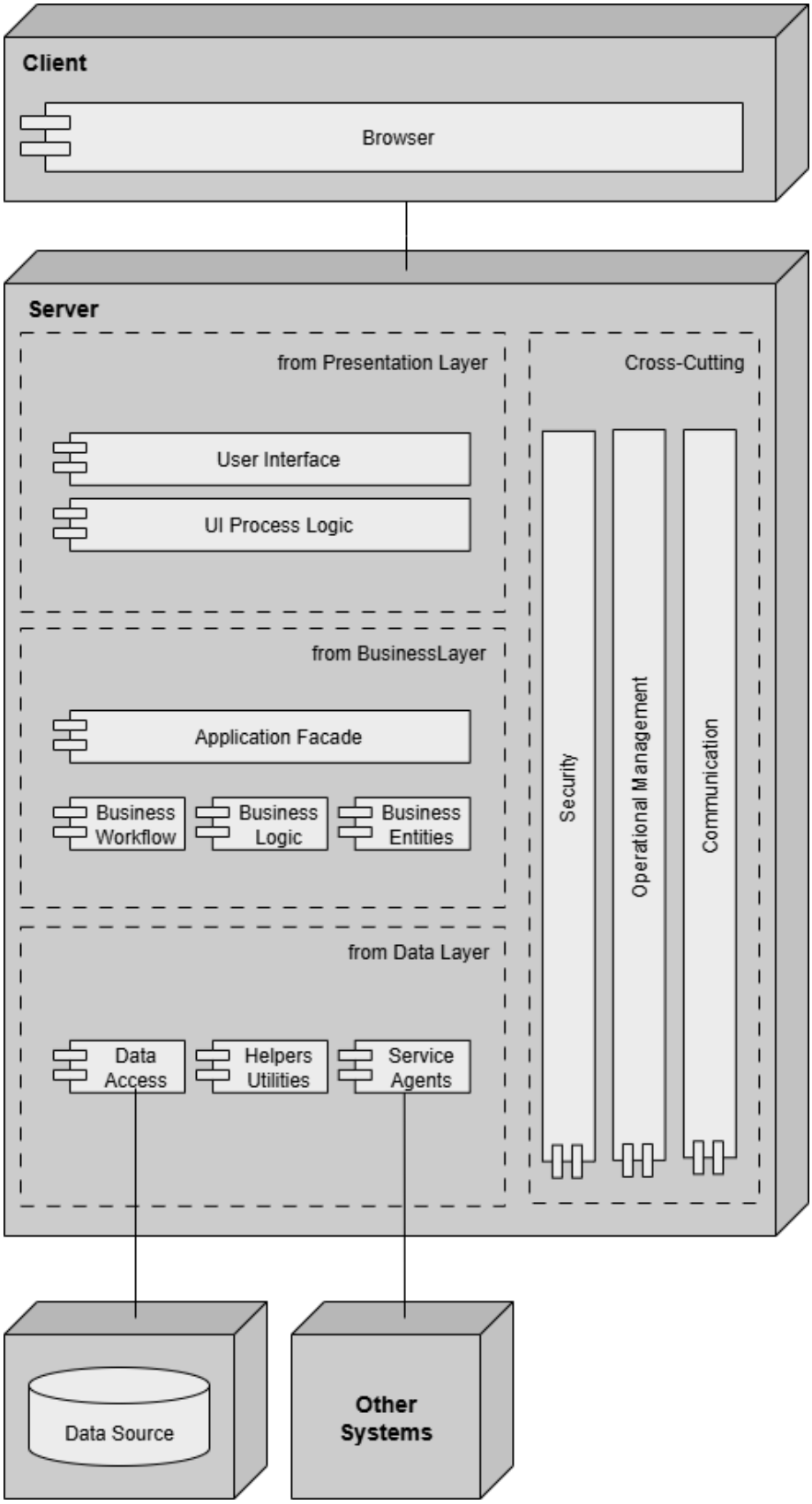


Figure A.2: Reference Architektur (Source: Cervantes and Kazman, 2016: 213)

References

Micah Altman and Richard Landau. 2024. Selecting Efficient and Reliable Preservation Strategies: Modeling Long-term Information Integrity Using Large-scale Hierarchical Discrete Event Simulation. *International Journal of Digital Curation* 18 (1): 1–24. (Cited on page 38).

AtMoDat. 2025. Atmospheric Model Data: Data Quality, Curation Criteria and DOI Branding. Accessed September 20, 2025. https://www.atmodat.de/. (Cited on page 15).

Aybüke Aurum and Claes Wohlin. 2003. The fundamental nature of requirements engineering activities as a decision-making process. *Inf Software Technology* 45 (14): 945–954. (Cited on page 17).

Michelle Barker, Neil P. Chue Hong, Daniel S. Katz, Mark Leggott, Andrew Treloar, Joris van Eijnatten, and Selina Aragon. 2021. Research software is essential for research data, so how should governments respond? Zenodo. (Cited on page 1).

Len Bass, Paul Clements, and Rick Kazman. 2022. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley. (Cited on pages 3, 19, 38 sq., 42).

Bausteine Forschungsdatenmanagement. 2025. Accessed July 2, 2025. https://bausteine-fdm.de/. (Cited on page 6).

Christopher Blech, Nils Dreyer, Matthias Friebel, Christoph Jacob, Mostafa Shamil Jassim, Leander Jehl, Rüdiger Kapitza, Manfred Krafczyk, Thomas Kürner, Sabine Christine Langer, Jan Linxweiler, Mohammad Mahhouk, Sven Marcus, Ines Messadi, Sören Peters, Jan-Marc Pilawa, Harikrishnan K. Sreekumar, Robert Strötgen, Katrin Stump, Arne Vogel, and Mario Wolter. 2022. SURESOFT: Towards Sustainable Research Software [in en]. (Braunschweig), (cited on page 29).

BMFTR. 2018. Bekanntmachung. Accessed September 20, 2025. https://www.bmftr.bund.de/SharedDocs/Bekanntmachungen/DE/2018/06/1791_bekanntmachung.html. (Cited on page 15).

———. 2025. Forschungsdatenmanagement. Accessed July 2, 2025. https://www.bmftr.bund.de/DE/Forschung/Wissenschaftssystem/Forschungsdaten/forschungsdaten_node.html. (Cited on pages 6, 11).

Büttner, Stephan, Hobohm, Hans-Christoph, and Müller, Lars, eds. 2011. Handbuch Forschungsdatenmanagement. (Cited on page 6).

Humberto Cervantes and Rick Kazman. 2016. Designing Software Architectures: A Practical Approach. SEI Series in Software Engineering. Addison-Wesley. (Cited on pages 3, 17, 19 sq., 22 sq., 35, 38, 41 sq., 47 sq., 50, 57, 63, 74, 81 sq.).

M. Chemuturi. 2012. Requirements Engineering and Management for Software Development Projects. New York: Springer. (Cited on page 18).

Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh. 2013. Characterizing Architecturally Significant Requirements. *IEEE Software* 30 (2): 38–45. (Cited on page 43).

Neil P. Chue Hong, Daniel S. Katz, Michelle Barker, Anna-Lena Lamprecht, Carlos Martinez, Fotis E. Psomopoulos, Jen Harrow, Leyla Jael Castro, Morane Gruenpeter, Paula Andrea Martinez, Tom Honeyman, Alexander Struck, Allen Lee, Axel Loewe, Ben van Werkhoven, Catherine Jones, Daniel Garijo, Esther Plomp, Francoise Genova, Hugh Shanahan, Joanna Leng, Maggie Hellström, Malin Sandström, Manodeep Sinha, Mateusz Kuzak, Patricia Herterich, Qian Zhang, Sharif Islam, Susanna-Assunta Sansone, Tom Pollard, Udayanto Dwi Atmojo, Alan Williams, Andreas Czerniak, Anna Niehues, Anne Claire Fouilloux, Bala Desinghu, Carole Goble, Céline Richard, Charles Gray, Chris Erdmann, Daniel Nüst, Daniele Tartarini, Elena Ranguelova, Hartwig Anzt, Ilian Todorov, James McNally, Javier Moldon, Jessica Burnett, Julián Garrido-Sánchez, Khalid Belhajjame, Laurents Sesink, Lorraine Hwang, Marcos Roberto Tovani-Palone, Mark D. Wilkinson, Mathieu Servillat, Matthias Liffers, Merc Fox, Nadica Miljković, Nick Lynch, Paula Martinez Lavanchy, Sandra Gesing, Sarah Stevens, Sergio Martinez Cuesta, Silvio Peroni, Stian Soiland-Reyes, Tom Bakker, Tovo Rabemanantsoa, Vanessa Sochat, Yo Yehudi, and RDA FAIR4RS WG. 2022. FAIR Principles for Research Software (FAIR4RS Principles). V. 1.0, May. (Cited on pages 2, 16).

Lawrence Chung. 1998. Architecting Quality: A Goal-Oriented, Knowledge-Based Approach. *Proceedings KUST,* (cited on page 16).

Connie Clare. 2019. Engaging researchers with data management the cookbook. Edited by Maria Cruz and Elli Papadopoulou. 153. Open Reports series. OpenBook Publishers. (Cited on page 6).

Paul Clements, Rick Kazman, and Mark Klein. 2002. Evaluating Software Architectures. SEI Series in Software Engineering. Addison-Wesley. (Cited on page 23).

Conquaire. 2019. Accessed January 14, 2019. https://gitlab.ub.uni-bielefeld.de/conquaire. (Cited on page 15).

Data Science Journal. 2025. Accessed July 2, 2025. https://datascience.codata.org/. (Cited on page 6).

Alan M. Davis. 2003. The art of requirements triage. *IEEE Computer* 36 (3): 42–49. (Cited on page 17).

DMPOnline. 2025. Accessed July 2, 2025. https://dmponline.dcc.ac.uk. (Cited on page 26).

Stephan Druskat, Nasir U. Eisty, Robert Chisholm, Neil P. Chue Hong, Ryan C. Cocking, Myra B. Cohen, Michael Felderer, Lars Grunske, Sarah A. Harris, Wilhelm Hasselbring, Thomas Krause, Jan Linxweiler, and Colin C. Venters. 2025. Better Architecture, Better Software, Better Research. *Computing in Science & Engineering* 27 (2): 45–57. (Cited on pages 15, 69).

Stephan Druskat, Lars Grunske, Caroline Jay, and Daniel S. Katz. 2024. Research Software Engineering: Bridging Knowledge Gaps (Dagstuhl Seminar 24161). Edited by Stephan Druskat, Lars Grunske, Caroline Jay, and Daniel S. Katz. *Dagstuhl Reports* (Dagstuhl, Germany) 14 (4): 42–53. (Cited on page 14).

Peter Eeles and Peter Cripps. 2009. The Process of Software Architecting. Upper Saddle River: Addison-Wesley Professional. (Cited on page 19).

Michael Felderer, Michael Goedicke, Lars Grunske, Wilhelm Hasselbring, Anna-Lena Lamprecht, and Bernhard Rumpe. 2025. Investigating Research Software Engineering: Toward RSE Research. *Commun. ACM* (New York, NY, USA) 68, no. 2 (January): 20–23. (Cited on pages 1, 14 sq., 76).

forschungsdaten.info. 2025. Accessed July 2, 2025. https://forschungsdaten.info. (Cited on pages 6, 11).

Heather Greer Klein, Gustavo Durand, Rory McNicholl, Arron Griffith, Alex Ioannidis, Kirsta Stapelfeldt, Kristi Park, Donald Moses, and Kate Dohe. 2024. The Repository Rodeo. Open Repositories 2024 (OR2024), Gothenburg, Sweden. Zenodo. (Cited on page 27).

GREI. 2025. Generalist Repository Ecosystem Initiative. Accessed September 20, 2025. https://datascience.nih.gov/data-ecosystem/generalist-repository-ecosystem-initiative. (Cited on page 15).

Harvard Data Science Review. 2025. Accessed July 2, 2025. https://hdsr.mitpress.mit.edu/. (Cited on page 6).

Wilhelm Hasselbring, Stephan Druskat, Jan Bernoth, Philine Betker, Michael Felderer, Stephan Ferenz, Ben Hermann, Anna-Lena Lamprecht, Jan Linxweiler, Arnau Prat, Bernhard Rumpe, Katrin Schöning-Stierand, and Shinhyung Yang. 2025. Multidimensional Research Software Categorization. *Computing in Science & Engineering* 27 (2): 59–68. (Cited on pages 13 sqq., 76).

Hanna Hedeland and Anne Ferger. 2020. Towards Continuous Quality Control for Spoken Language Corpora. *International Journal of Digital Curation* 15 (1): 1–13. (Cited on page 15).

Kerstin Helbig and Janna Neumann. 2016. Lehrbuch Forschungsdatenmanagement. Accessed July 2, 2025. https://vivo.tib.eu/fis/individual/smb201611471147. (Cited on page 6).

Patrick Helling. 2025. Entwicklung eines formalen Beschreibungsmodells für das geisteswissenschaftliche Forschungsdatenmanagement - eine qualitative Untersuchung von Beratungsprotokollen zur bedarfsorientierten Beschreibung, Strukturierung und Modellierung des Managements von digitalen Forschungsdaten. PhD diss., Universität zu Köln, November. (Cited on page 25).

Simon Hettrick. 2016. Research Software Sustainability Report on a Knowledge Exchange Workshop. The Software Sustainability Institute. (Cited on page 15).

Simon Hettrick, Mario Antonioletti, Les Carr, Neil Chue Hong, Stephen Crouch, David De Roure, Iain Emsley, Carole Goble, Alexander Hay, Devasena Inupakutika, Mike Jackson, Aleksandra Nenadic, Tim Parkinson, Mark I. Parsons, Aleksandra Pawlik, Giacomo Peru, Arno Proeme, John Robinson, and Shoaib Sufi. 2014. UK Research Software Survey 2014. [Data set] Zenodo. (Cited on page 25).

Hubert Hofmann and Franz Lehner. 2001. Requirements engineering as a success factor in software projects. *IEEE Software* 18 (4): 958–66. (Cited on page 17).

Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, and Pierre America. 2007. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software* 80 (1): 106–126. (Cited on page 19).

Christine Hofmeister, Robert L Nord, and Dilip Soni. 2005. Global analysis: moving from software requirements specification to structural views of the software architecture. *IEE Proceedings-Software* 152 (4): 187–197. (Cited on page 18).

Colin Hood, Simon Wiedemann, Stefan Fichtinger, and Urte Pautz. 2008. Requirements management: The interface between requirements development and all other systems engineering processes. Springer Science & Business Media. (Cited on page 18).

Invenio. 2025a. Curations. Accessed September 20, 2025. https://github.com/tu-graz-library/invenio-curations. (Cited on page 15).

———. 2025b. RDM. Accessed September 20, 2025. https://inveniosoftware.org/blog/2023-05-30-may-curation-validation/. (Cited on page 15).

ISO. 1994. Accessed July 2, 2025. https://www.iso.org/standard/20269.html. (Cited on page 32).

ISO. 2023. Accessed July 2, 2025. https://www.iso.org/standard/78176.html. (Cited on pages 38, 42).

Arne Johanson and Wilhelm Hasselbring. 2018. Software Engineering for Computational Science: Past, Present, Future. *Computing in Science & Engineering* 20 (2): 90–109. (Cited on pages 1 sq., 13 sqq.).

Daniel S. Katz, Jeffrey C. Carver, Neil P. Chue Hong, Sandra Gesing, Simon Hettrick, Tom Honeyman, Karthik Ram, and Nicholas Weber. 2021. Addressing Research Software Sustainability via Institutes. In *2021 IEEE/ACM International Workshop on Body of Knowledge for Software Sustainability (BoKSS),* 11–12. (Cited on pages 15, 28).

Daniel S. Katz, Tom Honeyman, Paula Andrea Martinez, Michelle Barker, Leyla Jael Castro, Neil Chue Hong, Morane Gruenpeter, Jennifer Harrow, Anna-Lena Lamprecht, Carlos Martinez-Ortiz, and Fotis Psomopoulos. 2022. FAIR for Research Software (FAIR4RS): A summary, August. (Cited on page 16).

Inna Kouper and Gretchen R. Stahlman. 2025. Two Decades, Same Story? Insights and Future Directions in Long Tail Data Curation. *International Journal of Digital Curation* 19 (1): 1–11. (Cited on page 38).

Till Kreutzer and Henning Lahmann. 2019. Rechtsfragen bei Open Science: Ein Leitfaden. Hamburg University Press. (Cited on page 12).

Philippe Kruchten. 2003. The Rational Unified Process: An Introduction. Upper Saddle River: Addison-Wesley Professional. (Cited on page 19).

Kruse, Filip and Thestrup, Jesper Boserup, eds. 2018. Research Data Management - A European Perspective. Berlin, Boston: De Gruyter Saur. (Cited on page 6).

Anna-Lena Lamprecht, Bala Desinghu, Christian Busse, Daniel Garijo, Esther Plomp, Franck Giacomoni, Jen Harrow, Javier Molden, Katherine Johnston, Leyla Jael Garcia Castro, Margareta Hellstrom, Michelle Barker, Natalie Meyers, Neil Philippe Chue Hong, Paula Andrea Martinez, Patricia Herterich, Qian Zhang, Sandra Gesing, Sergio Martinez Cuesta, and Tom Honeyman. 2021. What makes software FAIR? Reviewing the Towards FAIR Principles for Research Software paper and new research related to FAIR software: A report from FAIR4RS Subgroup 4. V. 0.1, June. (Cited on page 16).

Gitte Lindgaard, Richard Dillon, Patricia Trbovich, Rachel White, Gary Fernandes, Sonny Lundahl, and Anu Pinnamaneni. 2006. User Needs Analysis and requirements engineering: Theory and practice. *Interacting with Computers 18* 18 (2006): 47–70. (Cited on page 19).

Walid Maalej and Anil Kumar Thurimella. 2013. An Introduction to Requirements Knowledge. In *Managing Requirements Knowledge,* edited by Walid Maalej and Anil Kumar Thurimella, 1–20. SpringerLink. Berlin: Springer. (Cited on pages 17 sq.).

Robert C. Martin. 2018. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall. (Cited on pages 48, 78).

Microsoft. 2009. Application Architecture Guide. Microsoft Press. (Cited on page 19).

Barend Mons. 2018. Data Stewardship for Open Science: Implementing FAIR Principles. Chapman / Hall. (Cited on page 6).

NFDI. 2025a. Accessed July 2, 2025. https://base4nfdi.de/projects/dmp4nfdi. (Cited on page 26).

NFDI. 2025b. Accessed July 2, 2025. https://base4nfdi.de/projects/jupyter4nfdi. (Cited on page 27).

Bashar Nuseibeh and Steve Easterbrook. 2000. Requirements engineering: a roadmap. In *Proceedings of the conference on the future of software engineering (ICSE'00),* 35–46. New York: ACM. (Cited on page 17).

Hagen Peukert, Thomas Asselborn, Ralf Möller, and Sylvia Melzer. 2025. Data Curation as a Stepwise Service to Data Sustainability: The Grey Area Between Small-Scale Applications and Large-Scale Data Repositories. In *2025 IEEE International systems Conference (SysCon),* 1–5. (Cited on page 38).

Klaus Pohl. 1996. Process-centered requirements engineering. New York: Wiley. (Cited on page 17).

Klaus 1960- Pohl. 2010. Requirements engineering fundamentals, principles, and techniques. Springer. (Cited on page 3).

Putnings, Markus, Neuroth, Heike, and Neumann, Janna, eds. 2021. Praxishandbuch Forschungsdatenmanagement. Berlin, Boston: De Gruyter Saur. (Cited on page 6).

QUEST. 2025. Quality - Established: Erprobung und Anwendung von Kurationskriterien und Qualitätsstandards für audiovisuelle, annotierte Sprachdaten. Accessed September 20, 2025. https://www.slm.uni-hamburg.de/ifuu/forschung/forschungsprojekte/quest.html. (Cited on page 15).

RDMO. 2025a. Accessed July 2, 2025. https://rdmo.aip.de. (Cited on page 26).

RDMO. 2025b. Accessed July 2, 2025. https://rdmorganiser.github.io. (Cited on page 26).

RfII. 2025. Leistung in Verantwortung. Zur Zukunft der wissenschaftlichen Informationsinfrastrukturen in Deutschland. Göttingen. (Cited on page 1).

Mark Richards and Neal Ford. 2025. Fundamentals of Software Architecture. O'Reilly. (Cited on pages 48, 50, 57, 62 sq., 66, 68, 73, 78).

Matthew Skelton and Manuel Pais. 2024. Team Topologies Organisation von Business- und IT-Teams für einen schnellen Arbeitsfluss : Inklusive Interaktionen für verteilte Teams - Workbook : Team-Topologies-Patterns für eine produktivere Zusammenarbeit. O'Reilly. (Cited on page 73).

Ian Sommerville. 1998. Software engineering. International computer science series. Addison-Wesley. (Cited on page 3).

Ian Sommerville and Pete Sawyer. 1997. Requirements engineering: a good practice guide. New York: Wiley. (Cited on page 17).

Jürgen Streicher, Marlies Schütz, Clemens Blümel, and Alexander Schniedermann. 2025. Open Research Tools: Stand und Herausforderungen im Spannungsfeld von Forschungssoftware, Offenheit und digitaler Infrastruktur, June. (Cited on pages 25, 28).

Surveys. 2025. Accessed July 2, 2025. https://www.forschungsdaten.org/index.php/Umfragen_zum_Umgang_mit_Forschungsdaten_an_wissenschaftlichen_Institutionen. (Cited on page 25).

FAIR4RS WG. 2021. FAIR4RS Subgroup 4 - reading list of new research. V. 1.0, February. (Cited on page 16).

Michael W. Whalen, Andrew Gacek, Darren Cofer, Anitha Murugesan, Mats P.E. Heimdahl, and Sanjai Rayadurgam. 2013. Your "What" Is My "How": Iteration and Hierarchy in System Design. *IEEE Software* 30 (2): 54–60. (Cited on page 17).

Wiki Forschungsdaten. 2025. Accessed July 2, 2025. https://www.forschungsdaten.org/. (Cited on page 6).

Kai Joachim Wörner. 2015. Auswertung der Professorenumfrage zum Konzept eHumanities 2020+. (Cited on page 28).

Stephan Wünsche, Volker Soßna, Vanessa Kreitlow, and Pia Voigt. 2022. Urheberrechte an Forschungsdaten – Typische Unsicherheiten und wie man sie vermindern könnte: Ein Diskussionsimpuls. *Bausteine Forschungsdatenmanagement,* no. 1 (March): 26–42. (Cited on page 12).

Da Yang, Di Wu, Supannika Koolmanojwong, Windsor A. Brown, and Barry W. Boehm. 2008. Wikiwinwin: a wiki based system for collaborative requirements negotiation. In *Proceedings of the HICCS,* 24. Waikoloa. (Cited on page 17).

List of Abbreviations

ACDM . . . . . Architecture-Centric Design Method
ADD . . . . . Attribute-Driven Design
ADP . . . . . Acyclic Dependency Principle
AI . . . . . Artificial Intelligence
API . . . . . Application Programming Interface
ASC . . . . . Architecture Separation of Concerns
ASR . . . . . Architecturally Significant Requirement
ATAM . . . . . Architecture Tradeoff Analysis Method
BAPO . . . . . Business Architecture Process and Organization
CCP . . . . . Common Closure Principle
CON . . . . . Constraint
CRN . . . . . Concern
CRP . . . . . Common Reuse Principle
DIP . . . . . Dependency Inversion Principle
DMP . . . . . Data Management Plan
DMP4NFDI . . . Data Management Plan for Nationale Forschungsdateninfrastrukturen (National Research Data Infrastructure)
DOI . . . . . Digital Object Identifier
DTF . . . . . Data Transfer Facilitator
FAIR . . . . . Findable, Accessible, Interoperable, Reusable
FAIR4RS . . . . . Findable, Accessible, Interoperable, Reusable for Research Software
FCD . . . . . Final Committee Draft
FORGE . . . . . Forschungsdaten in den Geisteswissenschaften (Research Data in the Humanities)
IEC . . . . . International Electrotechnical Commission
ISO . . . . . International Organization for Standardization
ISP . . . . . Interface Segregation Principle
IT . . . . . Information Technology
LAE . . . . . Lightweight Architecture Evaluation
LSP . . . . . Liskov Substitution Principle
NFR . . . . . Non-functional Requirements
NoSQL . . . . . Not Only Structured Query Language
OCP . . . . . Open-Closed Principle
OSI . . . . . Open System Interconnection
PI . . . . . Principal Investigator
PID . . . . . Persistent Identifier

QA . . . . . Quality Attribute
QAW . . . . . Quality Attribute Workshop
RCA . . . . . Rich Client Application
RDLC . . . . . Research Data Life Cycle
RDM . . . . . Research Data Management
RE . . . . . Requirements Engineering
REP . . . . . Reuse/Release Equivalence Principle
RM . . . . . Requirements Management
RIA . . . . . Rich Internet Application
RSE . . . . . Research Software Engineering
RUP . . . . . Rational Unified Process
SAP . . . . . Stable Abstractions Principle
SD . . . . . Software Development
SDP . . . . . Stable Dependency Principle
SE . . . . . Software Engineering
SOA . . . . . Service Oriented Architecture
SOLID . . . . . Acronym for SRP, OCP, LSP, ISP, DIP
SP . . . . . Service Provider
SQL . . . . . Structured Query Language
SRP . . . . . Single Responsibility Principle
SSO . . . . . Single Sign On
TLR . . . . . Technology Readiness Level
UC . . . . . Use Case
UML . . . . . Unified Modeling Language

Index