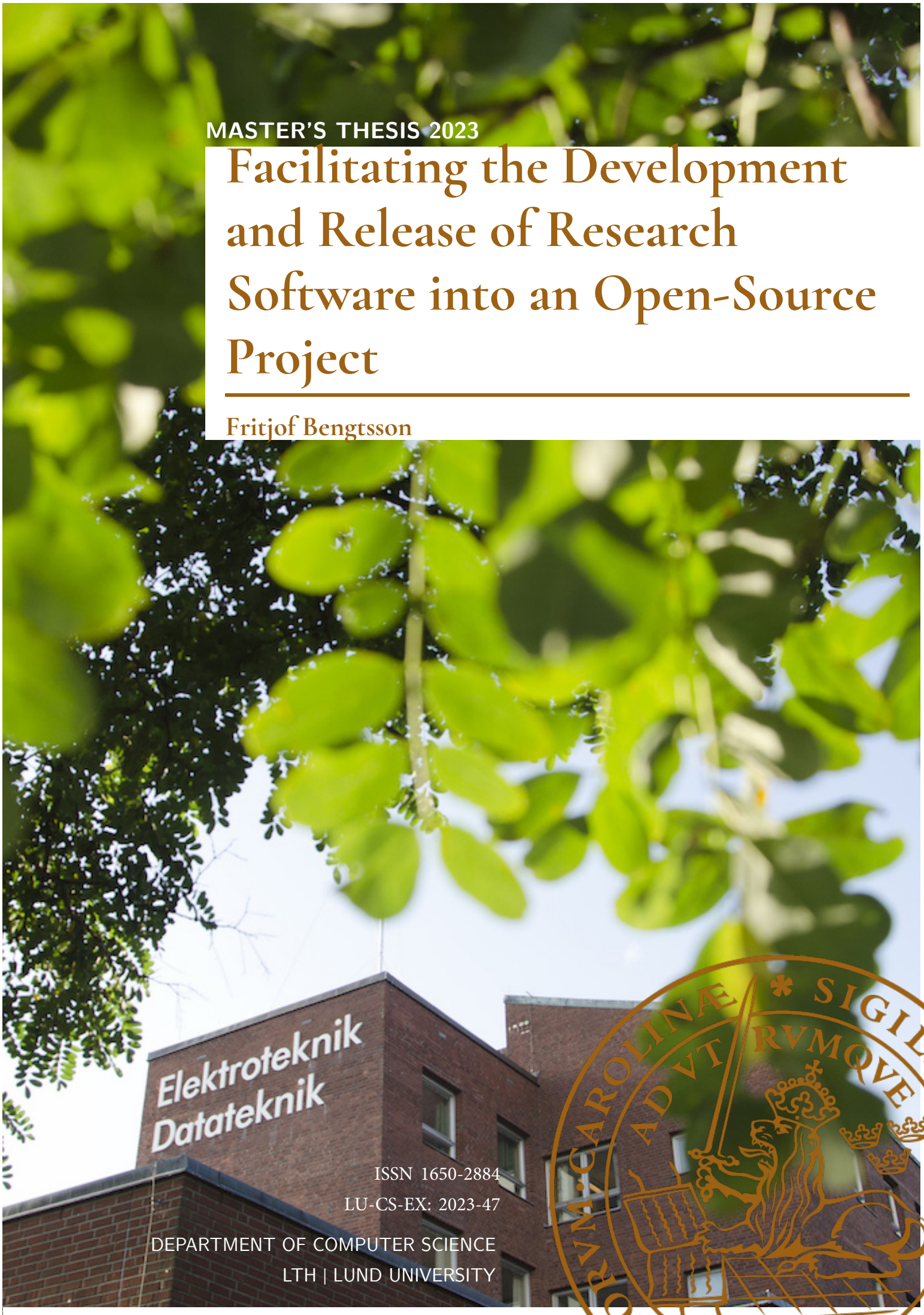

MASTER'S THESIS 2023

Facilitating the Development and Release of Research Software into an Open-Source Project

Fritjof Bengtsson

ISSN 1650-2884
LU-CS-EX: 2023-47

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-47

Facilitating the Development and Release of Research Software into an Open-Source Project

Främjande av utvecklandet och utgivandet av forskningsprogramvara som öppen källkod

Fritjof Bengtsson

Facilitating the Development and Release of Research Software into an Open-Source Project

Fritjof Bengtsson
eko15fbe@student.lu.se

November 7, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors:
Alexander Ekman, alexander.ekman@hep.lu.se
Alma Orucevic-Alagic, alma.orucevic-alagic@cs.lth.se
Caterina Doglioni, caterina.doglioni@cern.ch

Examiner: Martin Höst, martin.host@cs.lth.se

Abstract

This work presents an account of the process of transforming Baler, a data compression tool for scientific data, into an open-source software project. Initially created for specialized scientific use, the expansion into an open-source project required combining the expertise of various scientific fields.

Navigating this process presented several challenges, including managing a steep learning curve for scientists new to the industry standard of software development, deciding on a license, and addressing technical issues with limited existing solutions.

The transition of Baler highlights the potential of open-source approaches in scientific software development. The interdisciplinary collaboration established during the process marks the value of combining diverse knowledge and skills. This experience provides valuable insights for similar projects and emphasizes the importance of open-source development in broadening the accessibility of scientific software tools.

Keywords: MSc, Open-Source, Compression, License, Software Development, Particle-physics, Design Science

Acknowledgements

The successful completion of this thesis would not have been possible without the support and guidance of several individuals. I am deeply grateful to my supervisor at the Department of Computer Science, Alma Orucevic-Alagic, for her invaluable insights and guidance on the thesis and open-source.

I also extend my sincerest thanks to Martin Höst, my examiner at the Department of Computer Science, for his expertise on open-source and his contributions to broadening my understanding of the topic.

Additionally, I would like to express my gratitude to Alexander Ekman, at the Department of Physics, for his tireless efforts on the Baler project. His dedication and passion for his work have been truly inspiring, especially his curiosity and eagerness to adopt good computer science and open-source standards.

I would also like to mention Axel Gallén from the Department of Physics at Lund University, and Pratik Jawahar from the Department of Physics at Manchester University. It has been a joy working together with both of you.

Contents

Chapter 1

Introduction

Particle physics is a branch of physics that studies the fundamental constituents of matter and their interactions. The Large Hadron Collider (LHC) at CERN[1] is the largest particle accelerator facility in the world, designed to produce high-energy particle collisions to explore the fundamental structure of matter and the universe. The LHC experiments generates vast amounts of data, that was essential in discoveries such as the Higgs boson [2] and insights into dark matter [3][4].

However, the sheer volume of data produced by the LHC experiments presents significant challenges for data processing and storage. The upgrade to the High Luminosity LHC in 2029 will increase the production of particle collisions ten-fold and in turn data generation five-fold [5][6]. This makes it imperative to only save the most relevant and valuable data for the scientific goals of the experiments. In response to these challenges, the Physics Department and Computer Science Department at Lund University, in collaboration with the Physics Department at Manchester University have developed a compression tool. The tool – Baler – utilizes lossy machine learning-based compression [7] to compress multi-dimensional data and evaluate the accuracy of the process.

Lossy machine learning-based compression has the potential to be a suitable method for compressing multi-dimensional data in the context of particle physics, as it could offer a balance between accuracy and statistical power. While it is important to maintain accuracy of the decompressed data in relation to the original dataset, the main focus is on maximizing the statistical power. Statistical power refers to the ability to detect a true effect or relationship within the data, which is usually enhanced by having a larger sample size. Therefore, the goal is to obtain the greatest possible statistical power by making use of a larger amount of data, while still ensuring the decompressed data remains accurate and representative of the original dataset. The goal of Baler is to enable researchers to efficiently compress and store more data without compromising the overall quality and usefulness of the information.

1.1 Motivation

In the context of handling large volumes of scientific data, Baler has the potential to tackle this challenge of efficient data compression, while also providing the capability to evaluate accuracy of the process. This ensures the scientific goals of the experiments are not compromised by the compression or decompression process. A goal of the Baler project is to enable researchers from vastly different fields to test feasibility of compressing different types of scientific data.

Making Baler an open-source tool has the potential to greatly expand its impact and reach. Open-source tools are not only freely available, but they also foster collaboration and promote the sharing of ideas and knowledge within the community [8]. By releasing Baler on the European Open Science Cloud (EOSC) platform [9], researchers in a variety of fields will have access to Baler. Our objective is to determine the strategy for making Baler an open-source tool and to facilitate its release on the EOSC platform, where it can be shared with researchers beyond the field of particle physics [9].

1.2 Objectives

The objective of this Master's thesis is to develop a strategy for transitioning Baler into an open-source environment, so it can be released on the EOSC platform and made widely available to researchers in various fields. To achieve this objective, we will address the following research questions:

1. How can Baler be designed to meet the data requirements of different fields of science and industry?

2. How can we establish and foster an open-source community around the Baler tool, and what are the key elements for creating a sustainable open-source project?

3. What metrics can be used to assess adoption and impact of Baler?

By answering these research questions, we aim to determine a product road-map for Baler that supports its use in a variety of data-intensive scientific applications, and to establish a strategy for making Baler an open-source tool that can be used and improved upon by researchers in these fields.

1.3 Limitations

This section details the limitations of the results presented in this thesis. The research conducted has produced useful findings. However, it is necessary to be aware of certain limitations of our research. These don't take away from our work's value but are important to bear in mind when interpreting the results. These limitations also point towards areas where more research could be beneficial, particularly in managing open-source project for scientific software development.

1.3.1 Isolated project

In this section, we present a key limitation of the case study, namely, the isolated nature of the Baler project. While our investigation has provided valuable insights into the challenges and best practices associated with managing an open-source project with non-developer contributors (individuals who contribute to the project in roles other than software development or are coming from a background with limited exposure to software development), it is important to recognize that these findings are based on the analysis of a single project.

The experiences and outcomes observed in the Baler project may not necessarily be generalizable to other open-source or research software projects, particularly those with different scopes, goals, or team compositions. While the lessons learned from the Baler project can potentially offer guidance for future projects, it is crucial to acknowledge that the results presented in this study may not be universally applicable.

To overcome the fact that the insights from the Baler project might not be universally applicable, we suggest a future study involving a survey. This survey should be aimed at scientists across various fields, such as physics, biology, and beyond, who are actively developing, or have developed, software for their research. The goal of this survey would be to gather a variety of perspectives on the challenges experienced during the development of scientific software, particularly those transitioning to open-source. This would enable a comparison between the Baler project and those found in the broader scientific context.

Such research would enable a more comprehensive understanding of the factors influencing the success and efficiency of open-source projects involving non-developer contributors, ultimately contributing to more effective project management and collaboration.

1.3.2 Not about machine learning

Another limitation of this study lies in its primary focus on the process of making Baler an open-source project and fostering a healthy open-source community with scientists, rather than delving into the technical details of the autoencoder implementation. This study does not offer an in-depth analysis of the technical aspects of the Baler implementation.

As a result, readers interested in the specifics of the machine learning techniques employed in Baler might not find the desired level of detail in this study. This limitation also implies that the present research may not directly contribute to the advancement of machine learning-based compression methods or their applications.

It is important to recognize that the scope of this study is intentionally focused on the open-source community and collaboration aspects, as the primary goal is to explore the unique challenges and opportunities associated with managing an open-source project involving non-developer contributors. For those seeking a more comprehensive understanding of the implementation of Baler, we suggest reading our paper Baler–Machine Learning Based Compression of Scientific Data [10], which delves into the machine learning compression techniques and their performance. Details of its implementation and performance in particle physics can found in An Open-Source Autoencoder Compression Tool for High Energy Physics by Axel Gallén [11]. We also suggest reading the article Deep Autoencoders for Compression in High Energy Physics [12] by Eric Wulff, to which some extent the Baler project is based on. Finally, we suggest delving into the source code [7][13].

Chapter 2
Methodology

This research primarily utilizes a case study methodology [14] to probe the challenges and best practices in open-source software development, especially concerning the collaboration between developers and non-developers. While the literature study [15] provides the foundational theoretical framework and academic rigor to the research, the case study remains the centerpiece, offering a rich and nuanced exploration of the intricacies inherent in open-source development.

The design science approach [16] is integrated as a supplementary method, serving a dual purpose. On one hand, it aids in problem-solving during the case study, ensuring practical issues encountered are aptly addressed with innovative solutions. On the other, it places a firm emphasis on the generation of technological rules stemming from the insights garnered, enhancing the applicability of the research findings. This approach accentuates the contributions' novelty, their tangible relevance to the real world, and their alignment with scientific rigor.

In essence, the symbiotic blend of a case study with supportive methodologies, like the literature review and design science, ensures that this research stands as both an academic exploration and a practical guide, poised to offer actionable insights and recommendations for future projects in the domain of open-source software development.

2.1 Literature Study

This section outlines the sources and the approach used in gathering existing information relevant to the study. A mix of books and academic articles were consulted to provide a foundational understanding and context. The subsequent subsections provide a look at the resources chosen and the rationale behind their selection.

2.1.1 Database Choice

This research primarily employed Google Scholar for its extensive reach across various disciplines. However, due to concerns over quality control and unclear indexing guidelines [17], Scopus was also utilized. Recognized for its rigorous selection criteria and refined search capabilities, Scopus served as a more robust and precise complement to the broader scope of Google Scholar [18], ensuring comprehensive foundation for the literature study.

2.1.2 Theoretical Foundation on Open-Source

A considerable portion of the research into open-source was grounded in two primary texts:

1. **Proudcing Open Source Software** by Karl Fogel [8]. This work offers a detailed exploration of the mechanisms, practices, and challenges of open-source software production

2. **The Cathedral and the Bazaar** by Eric Raymond [19]. Raymond discusses contrasting methodologies in software development, presenting insights into open-source practices.

These books set the stage for the theoretical framework adopted in the research. The literature sourced from Google Scholar was employed to either support, nuance, or contest the views presented by Fogel and Raymond. This range of literature enriches the theoretical depth of the study; it also ensures a balanced and rounded discourse.

Google Scholar Search Strategy

Given the quality control issues on Google Scholar mentioned earlier, a conscious effort was made to narrow down the results to those published by reputed institutions such as the *Institute of Electrical and Electronics Engineers* (IEEE) [20]. The search strings primarily followed the pattern "open-source <TOPIC>". As an example, when investigating the realm of security in open-source, the query "open-source security" was used.

Given the absence of advanced sorting mechanisms on Google Scholar, the default "Sort by relevance..." setting was used. The literature sourced from these searches was predominately incorporated in chapter 3–Background–providing depth, support, and occasional counterpoints to the foundational theories drawn from Fogel and Raymond.

Case Study Research

To gain insights into the methodology of case studies within the software engineering realm, a Scopus advanced search was conducted. The search string used specifically was: "case study" AND "software engineering" AND (LIMIT-TO (AF-ID , "The University of Manchester" 60003771) OR LIMIT-TO (AF-ID , "Lunds Universitet" 60029170)) AND (LIMIT-TO (SUBJAREA , "COMP")). The result was sorted by number of citations in descending order. This criterion prioritized works affiliated with Lund University and Manchester University in the context of computer science, given the collaborative nature of the Baler project between these institutions.

The first five works yielded by the search:

1. **Experimentation in Software Engineering** by Wohlin, C., Runeson, P., Höst, M., Regnell, B., Wesslén, A., 2012 [21]

2. **Guidelines for conducting and reporting case study research in software engineering** by Runeson, P., Höst, M., in Empirical Software Engineering, 2009 [14].

3. **Case Study Research in Software Engineering: Guidelines and Examples** by Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012 [22]

4. **Information systems and developing countries: Failure, success, and local improvisations** by Heeks, R. in Information Society, 2002 [23]

5. **Detection of duplicate defect reports using natural language processing** by Runeson, P., Alexandersson, M., Nyholm, O., in Proceedings - International Conference on Software Engineering, 2007 [24]

Of the results, the first three works were of particular interest. Their focus on case study methodologies, combined with the high citation counts, underline their significance in shaping contemporary approaches and understanding in software engineering case studies.

2.2 Case Study

The case study component of this research focuses on the Baler project, offering an in-depth look into the real-world application of the concepts and best practices identified from the literature. The case study is based on the guidelines presented by Runeson and Höst in their paper [14]. This section provides an overview of the case study methodology and delves into the Baler project as the case. An overview of the case study can also be seen in Table 2.1. The results derived from this case study are presented in Chapter 4–Results and Discussion–of this paper. We also present technical rules–guidelines–in Chapter 5.

2.2.1 Objective and Purpose

As the nature of scientific inquiry has evolved, so too have the tools that support it. Central to this evolution is the burgeoning domain of scientific software. While the creation of tools like Baler–previously introduced in Chapter 1–represents advancements, ensuring their adoption, adaptation, and widespread use requires another layer of strategy. This is particularly relevant when considering the rich collaborative potential of transitioning such tools to an open-source platform.

Primary Objective: The main focus of this case study is to understand the intricate processes, challenges, and strategies associated with transitioning Baler to an open-source platform. As presented in section 1.3, our insights are rooted in a single, albeit comprehensive, case study of Baler.

Collaboration Dynamics: An embedded objective within this primary aim is to explore the collaborative dynamics between developers and non-developers. This interaction is pivotal in open-source projects, especially when the contributors hail from diverse scientific fields with varying levels of software development experience. Drawing insights from the Baler project will offer a lens into the complexities of such interdisciplinary collaborations

Table 2.1: Overview of the Baler Case Study

Aspect	Description
Case and Units (2.2.3)	• Primary Case: Baler project. • Units: Developer's diary, interactions with researchers.
Objectives, Questions, and Hypotheses (1.2, 2.2.1)	• Transition understanding. • Collaboration dynamics exploration. • Actionable guidelines derivation.
Theoretical Basis (2.2.4)	• Grounded in literature, particularly Fogel's works.
Authors' Intentions (2.2.1)	• Examine Baler transition. • Define open-source transition guidelines. • Seed broader research.
Case Definition (1, 2.2.2)	• Focus: Baler transition. • Concerns: IP, quality, community. • Perspectives: Researchers of Baler.
Data Triangulation (2.2.3)	• Qualitative: Developer's diary, researcher interactions.
Rationale for Selection (2.2.1)	• Baler's representation of open-source transition.
Construct Validity (2.2.5)	• Baler-specific insights, broader aims.
Integrity Consideration (2.2.2)	• Focused on Baler researchers, with verbal agreements.

Deriving Guidelines: Stemming from our understanding of Baler's transition and collaborative dynamics, this study aims to derive actionable guidelines (technical rules, as presented in section 2.3 on design science [16]). These guidelines seek to aid research software projects in navigating their open-source journey, ensuring seamless transition, broad reach, and fostering healthy collaboration.

Broader Implications: While our findings are rooted in the Baler project, we acknowledge–as presented in section 1.3–that a single project's insights might not be universally applicable. However, our objective is to lay the foundation for broader research, possibly via future surveys and multiple case studies, to validate, refine, and expand upon the guidelines formulated here.

2.2.2 The Baler Project as Case

The Baler project, previously introduced in Chapter 1, signifies the journey of a research software project transitioning to open-source. This transition presents various challenges and opportunities, including intellectual property concerns, assurance of software quality, and fostering an open-source community.

At the heart of this exploration are the researchers actively involved with the Baler project. Their engagements, obstacles faced, and insights furnish a varied perspective on the multifaceted dynamics inherent in projects such as Baler.

To ensure the ethical integrity of this case study (as outlined in [14])–verbal agreements were established between us and the other researchers on the Baler project–granting permission to conduct the study on the project. These agreements served to clarify roles, responsibilities, and the handling of sensitive information. While the foundation of the study was built on trust, these measures were important in ensuring transparency and maintaining a working collaboration throughout the research process.

2.2.3 Data Collection Methods

For this study, data were gathered in two ways. The data is qualitative by nature. In turn, the case study on Baler becomes qualitative, even though it can be used for both qualitative and quantitative research [25]:

1. **Developer's Diary:** As a developer and researcher engaged with the Baler project, I maintained a diary. This diary became a chronicle of my journey in the project; encompassing observations, reflections, and interactions between myself and fellow researchers. It offers a look at both the technical challenges and the human narratives that paint the picture of transitioning research software into open-source.

2. **Interactions with Fellow Researchers:** While the diary primarily documents my personal experiences, it also integrates insights from interactions with other researchers. These interactions help broadening the scope of understanding and capturing a more holistic view of the project's ecosystem.

In line with the framework proposed by [14], our primary case is the Baler project itself. The diary entries and researcher interactions form the units of analysis.

2.2.4 Theoretical Framework

The methodological and theoretical underpinnings of this research draw deeply from the literature discussed in previous sections. While the structure and design of the Baler project case study are inspired by contemporary case study methodologies, the underlying principles of open-source software development are derived largely from Karl Fogel's insights.

Foundational Insights from Literature

Karl Fogel's book [8] serves as the primary theoretical compass for this research. Through his detailed examination of open-source development dynamics—spanning communication, version control, licensing, and conflict resolution—Fogel presents a holistic view of the practices and challenges inherent to open-source projects.

This comprehensive understanding, framed within Fogel's perspectives, is further enriched and contextualized using the case study methodologies identified in the Scopus search (2.1.2). The methodologies drawn from the likes of Wohlin [21], Runeson [14][24], and others are pivotal in shaping the research design, ensuring its alignment with accepted best practices in the field of software engineering.

2.2.5 Validity of the Baler Case Study

The Baler case study delves into transitioning research software to open-source. While its insights are valuable, it is imperative to evaluate the validity of the results, ensuring they accurately mirror real phenomena and are not skewed by research methodologies or the researchers' viewpoints. The four aspects of validity presented by Runeson and Höst [14]–as well as Yin [26]–are outlined below.

Construct Validity

The operational measures used in the Baler case study primarily concern the dynamics of open-source software development within a research-based context. A potential threat to construct validity emerges if these measures aren't consistently interpreted by both the researcher and the subjects of the study. However, continuous discussions were held within the project on a weekly basis to ensure alignment and shared understanding, thereby mitigating possible threats to construct validity.

Internal Validity

The study outlines various factors influencing the transition to open-source. An example of a potential third-party factor not accounted for could be the existing familiarity or expertise of the research team with open-source practices and tools. If some members of the team had prior experience, it might affect the transition speed and success rate, influencing the results.

External Validity

Given the specificity of the Baler case to a physics domain, generalizing findings to all research software realms can be ambitious. Yet, the broader strategies and challenges identified in transitioning to open-source might hold relevance across domains. Thus, while certain specifics may differ, the Baler case holds promise as a point of reference for future open-source transitions in diverse fields.

Reliability

Ensuring the reliability of the case study's results was of importance. While there was no formal review of collected data, in-depth discussions were held within the project concerning the researcher's observations and the experiences of the project's members. These discussions played a pivotal role in ensuring the alignment between the researcher's interpretations and the actual experiences of the subjects. However, the interpretation of these discussions were left to the researcher, who might have erroneously recalled the exact words or formulations. Because of this, we recommend–in alignment with the paper by Runeson and Höst [14]–that future research use a more systematic approach is used in terms of data collection and review.

2.3 Design Science

Design science offers a comprehensive lens for assessing and communicating research contributions [16].

Design science is a problem solving paradigm that seeks to create innovative solutions for real world problems. The key idea is that research must be both original and practical to contribute to the cumulative tradition of scientific knowledge. Therefore, it emphasizes the importance of producing a theoretical output – technological rules – that can guide practice and reflect on the relevance, novelty, and rigor of these rules.

2.3.1 Process

In line with design science methodology, this research is carried out in a series of iterative cycles that include problem identification, solution design, and evaluation phases.

Problem Identification: In this phase, a real-world problem is identified and its relevance is established. This phase involves an extensive literature review and consultation with experts in the field to fully understand the problem's context.

Solution Design: The next step involves the design of an innovative solution to the problem. The solution is developed based on a thorough understanding of the problem and the current state of the art in the field.

Evaluation: The final phase of each cycle involves evaluating the designed solution's performance and effectiveness. This is typically done through empirical studies, simulations, or other suitable methods.

The results from the evaluation feed into the next problem identification phase, creating an iterative cycle of continuous improvement.

2.3.2 Contribution

In terms of the contribution, this research not only aim to design and develop an innovative but also to generate technological rules. These rules, formulated based on the insights gained during the research process, serve as guiding principles for future application and further development.

Through this design science approach, this work aims to generate valuable insights and make meaningful contribution to the field between software engineering and science.

Chapter 3
Background

As modern scientific experiments produce an increasing amount of data, tools like Baler emerge as possible essential assets, streamlining data handling through machine learning-based compression techniques. In this chapter, we start by introducing Baler and touching on its foundational principle – autoencoders, highlighting their transformative potential for data-intensive scientific fields. Subsequent sections will touch on Baler's implementation using the `Python` [27] programming language and its use of `Docker` [28]. The chapter then moves on to provide an overview of open-source principles, with a specific emphasis on licensing.

3.1 Overview of Baler

Baler uses a neural network architecture called an "Autoencoder" [29] to reduce the size of multi-dimensional data whilst trained to have minimal discrepancy to the original data [7]. Autoencoders are composed of an encoder, a latent space that is smaller than the input, and a decoder. The encoder condenses information from the input into the latent space, while the decoder tries to reconstruct the original input from the compressed form. By storing the compressed data in the latent space along with the decoder network, the original data can be reconstructed when needed. Figure 3.1 gives a brief overview of how autoencoders and Baler works. Additionally, autoencoders can be used for anomaly detection [30]. Since any significant deviation from the training data, which results in a large reconstruction error, would be flagged as an anomaly [12]. The goal of Baler is to test the feasibility of compressing different types of scientific data using autoencoders. Tools like Baler can be crucial for enabling researchers to make the most of the data they can collect.

In addition to its potential impact in particle physics, the development and implementation of Baler have the potential to significantly impact other scientific disciplines and industries. By using machine learning to compress data, Baler offers an approach to data compression that has the potential to be applied in a wide range of contexts, such as computational

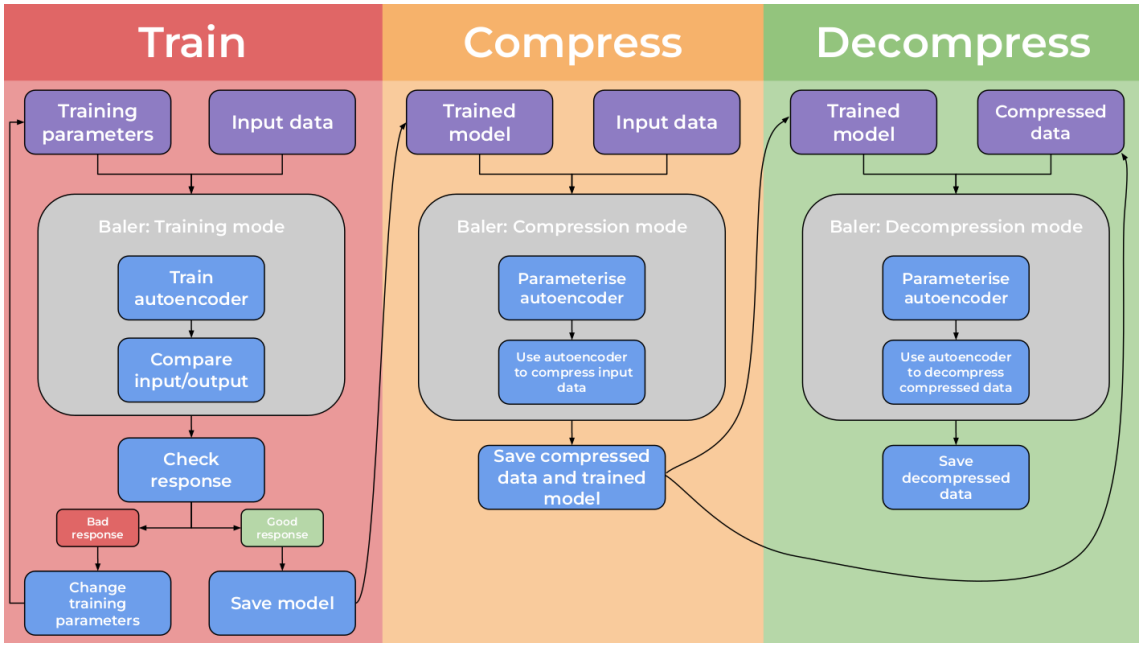

Figure 3.1: An overview of Baler's autoencoder implementation

fluid dynamics. This method has also shown to be promising on a number of different data sets in previous studies [31].

Overall, Baler could represent an advancement in the field of data compression and processing for scientific data, and its development and implementation could have a lasting impact on the field of particle physics, in addition to other areas of science and industry. However, for this to take place, it is crucial that a thriving community and continuous development is created around Baler.

3.1.1 Python

`Python` is a high-level, general-purpose programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum in the late 1980s [27], `Python` has become one of the most widely-used programming languages [32], thanks to its extensive standard library, numerous third-party packages, and active community support. Python supports multiple programming paradigms, including object-oriented, imperative, and functional programming, making it suitable for a diverse range of applications, such as web development, data analysis, artificial intelligence, and scientific computing [27].

In the Baler project, `Python` was chosen as the primary programming language due to its ease of use, flexibility, and the extensive machine learning libraries available (such as `PyTorch` [33] and `NumPy` [34]).

3.1.2 Docker

`Docker` is a tool that simplifies the process of developing, shipping, and running applications by using containerization technology [28][35]. Containers allow developers to package an application along with its dependencies, such as libraries and system tools, into a single,

portable unit. this approach ensures that the application runs consistently across various environments, eliminating the "it works on my machine" issue often encountered in software development.

`Docker` containers are lightweight, as they share the host system's kernel and isolate only the application and its dependencies [36]. This characteristic distinguishes `Docker` containers from traditional virtual machines [37][38], which require a full guest operating system to run. `Docker`'s containerization technology has become increasingly popular among developers [39], as it accelerates development cycles, simplifies deployment processes, and enhances application scalability.

In academia, and specifically in the field of software engineering, `Docker` has proven instrumental to promoting reproducibility in research, as it addresses many of the challenges associated with undocumented assumptions, dependencies, and configurations [40]. By packaging code and associated data into containers, `Docker` assists in overcoming barriers to the reproducibility of research artifacts, enhancing the reliability and credibility of scientific work.

In the context of the Baler project, distributing the software via a `Docker` image ensures platform independence and simplifies the installation process for users. By encapsulating the Baler application and its dependencies within a container, users can run the software seamlessly on any system with `Docker` installed, regardless of the underlying operating system.

3.2 Overview of Open-Source Software

Open-source software [8] represents a philosophy where the original source code is made freely available to the public. This allows anyone to view, modify, and distribute the software, fostering collaboration and transparency. The ethos behind open-source is not just about cost but emphasizes freedom: the freedom to share, adapt, and innovate. Prominent examples like the Mozilla Firefox browser [41] owe their origins to this principle, as they were built on open foundations where community contribution is central [8].

3.2.1 Contrasting Proprietary Software

Proprietary software is where the source code remains undisclosed and exclusive. This means only the original creators or authorized entities can access, modify, and distribute it [8]. Proprietary software often comes with licensing restrictions, limiting its use, distribution, and modification. Examples of this can be seen in many commercially available software products. Purchasing a license for such software allows you to use it, but you do not truly *own* it or have the liberty to tweak its underlying code or redistribute it as you wish. In contrast, while open-source software offers free access and customization advantages, some sectors still gravitate towards proprietary due to barriers in open-source adoption or strategies leveraged by proprietary vendors to maintain dominance [42].

While open-source platforms like Linux [43] often see a greater level of investment in applications than their counterparts such as Windows [44], the incentive dynamics for developers vary based on reputation effects and developer community size [45].

3.2.2 The Growth and Impact of Open-source

Recent years have witnessed a significant growth in open-source software's popularity and growth, challenging the norms of traditional software development [46]. While most studies previously focused on software built within conventional corporate structures, the rise of open-source projects, like the Linux operating system kernel [43], offers new perspectives. Despite its expansive nature, with over two millions lines of code, Linux [43] has consistently grown at a linear rate. This unexpected trajectory, diverging from tightly managed commercial software norms, highlights the dynamic potential and resilience of open-source platforms.

3.2.3 Benefits of Open-source Software

Open-source software catalyzed innovation, fostering a collaborative environment that spurred technological advancements and novel business models [47]. This movement reshaped software development paradigms, with methodologies like iterative development and agile techniques tracing their roots back to the open-source ethos [48]. Despite some criticism, the open-source model has shown resilience and innovation in the industry. Recent research [49] indicates that there is no substantial difference in quality between open-source and proprietary software.

Delving deeper into the benefits of open-source, the model inherently fosters a more diverse and inclusive environment for development [50]; this wide-ranging participation from developers not only speeds up the problem-solving process, but also introduces varied perspectives, often leading to holistic solutions.

Modern code review practices, as employed by industry giants like Google, further exemplify the robustness of open-source methodologies. In an exploratory investigation into modern code review at Google, it was identified that this practice was introduced early on and has been continually refined over the years, ensuring code quality and maintaining standards across millions of code reviews [51]. Such practices highlight the collaborative and rigorous nature of open-source development, where code is meticulously vetted, ensuring its reliability and robustness.

Another advantage is the heightened security [52][53]. With many eyes reviewing the code, vulnerabilities are detected and rectified swiftly, reducing the window of exploitation. Furthermore, the flexibility of open-source software ensures that it can be tailored to specific user requirements, making it particularly valuable for businesses looking to differentiate themselves or address niche needs. Additionally, it facilitates continuous learning and upskilling among developers [19]; by accessing and understanding the code behind major projects, new developers can gain insight and improve their coding skills.

Open-source software offers a unique approach to development, promoting transparency, collaboration, and flexibility. While not without its challenges [8][54], its merits stand firm against traditional proprietary models. As the digital landscape evolves, open-source remains an important consideration, reflecting the benefits of community-driven innovation. Its continued relevance emphasizes its value in the modern world.

3.2.4 Licensing

In the realm of open-source software, licenses form the legal foundation upon which community contributions thrive. These licenses dictate how software can be used, modified, and distributed, striking a balance between creative freedom and safeguarding intellectual property.

Key in this ecosystem are the Open Source Initiative (OSI) [55] and the Free Software Foundation (FSF) [56], two leading authorities that maintain comprehensive lists of open-source licenses. The OSI, renowned for its Open Source Definition [57], offers a seal of approval for licenses adhering to essential freedoms, ensuring standardization and compatibility within the community. Simultaneously, the FSF, through its commitment to software freedom, classifies licenses according to their adherence to the principles of free software, often highlighting the ethical and philosophical underpinnings of open-source development. Karl Fogel, in his writings on open-source software, emphasizes the intricate nature of these licenses and their crucial role in shaping the open-source ecosystem [8].

Licenses vary in their provisions. While some are permissive, granting developers wide latitude, others are restrictive, ensuring that the derived works retain the same level of openness [58][59]. Choosing a license is thus not merely a legal consideration but a declaration of the project's values and its stance on community collaboration [8].

The *General Public License (GPL)* stands out as a paragon of open-source licensing, often associated with the Linux operating system [43]. Under GPL, any derivative work based on the licensed software must also adopt the GPL, thereby perpetuating the spirit of openness [58].

Conversely, licenses like the *Apache License* [59] and the *MIT License* [60] offer a more permissive approach. These licenses permit users to use, modify, and distribute the software as they deem fit. Particularly, the Apache License provides the freedom without mandating the release of derivative works [59]. Similarly, the MIT License allows for unfettered use, modification, and distribution without any strings attached.

However, the world of open-source licensing isn't binary. Hybrid licenses, like the *Lesser General Public License (LGPL)*, carve a niche by combining elements of both open and proprietary systems. The LGPL, for instance, permits the use of open-source software within proprietary products, provided the open-source segment remains freely accessible [61].

Grasping the details of these licenses is important [62]. Each carries its own set of rules and permissions. As Fogel notes, understanding and adhering to these terms not only respect the creator's intent but also averts legal pitfalls [8]. As open-source continues to grow, navigating its licensing landscape with an informed mind mind becomes all the more crucial.

3.2.5 Measuring Success in an Open-Source Project

Open-source projects use many ways to measure their quality and success. Some studies have shown that most experts look at *code quality* to determine how good the software is, and they check *market success* and how active developers are to see how successful it is [63]. Another study observed 283 open-source projects for three years. They found that how popular a project is and how active its developers are can tell us a lot about its success. They also found that factors like the size of the user base, how man language versions there are, how responsibilities are given out, and how well the project is organized play a big role in its

success [64].

3.3 Open-Source within Academia

Open-source software holds significant value in academia, promoting collaboration and the free exchange of tools and knowledge for research advancement. It breaks away from the siloed development typical of proprietary systems (presented in section 3.2.1), encouraging diverse input and broad application.

Choosing Baler as a case study in the open-source context is motivated by practical and strategic factors. Baler's open-source nature invites a wider pool of knowledge and skill to further refine its utility and efficiency, potentially extending its relevance beyond its initial scope, such as particle physics.

The strategic aspect of what to share as open-source software is highlighted by Linåker et al. [65]. Similar to businesses, academic entities benefit from open innovation by leveraging external contributions and insights. Sharing Baler openly helps in harnessing diverse expertise for its development, thereby enhancing its capabilities and reliability.

Moreover, the importance of open-source software extends well beyond academia. Today, it is recognized as a crucial component in the digital infrastructure of various sectors, including the public sector where its adoption, although more recent, is on a significant rise [66]. Factors driving this demand include the potential for economic growth, innovation, and competition, alongside benefits particularly pertinent to the public sector context, such as improved interoperability, transparency, and digital sovereignty.

This trend is not limited to local or national initiatives. At an international level, organizations such as the World Bank and the United Nations are expressing interest in OSS, recognizing its potential to shape sectors critical for global development [66]. This shift underscores the need for comprehensive strategies in adopting and contributing to OSS, as it involves navigating complex procurement frameworks, legal constraints, and varying policy incentives unique to the public sector.

Furthermore, Baler serves as a tangible, accessible example of educational exploration, offering students and researchers a direct view into advanced neural network application. This open model ensures that Baler doesn't just remain a static tool but evolves with collective intelligence, possibly leading to breakthroughs in data processing or other scientific fields.

In summary, using Baler as an open-source case study underscores the importance of communal contribution in academia, potentially leading to enriched learning, improved research tools, and faster innovation [48]. The expanding adoption of open-source practices in the public sector reaffirms its value, suggesting a broader, more universal application that could revolutionize how governmental and non-governmental organizations operate and collaborate [66].

Chapter 4
Results and Discussion

The primary goal of this Master's thesis was to delve into the process of transitioning Baler to an open-source platform. This aim was motivated by the growing need for effective data compression and storage solutions, especially in data-rich fields such as particle physics.

To revisit the initial research questions:

1. How can Baler be designed to meet the data requirements of different fields of science and industry?
2. How can we establish and foster an open-source community around the Baler tool, and what are the key elements for creating a sustainable open-source project?
3. What metrics can be used to assess adoption and impact of Baler?

Our findings suggest:

1. Baler's design transitioned beyond mere technical solutions (4.2), evolving into a platform that meets diverse scientific needs through several strategic initiatives. Central to this was the establishment of a standardized data format to ensure consistency and compatibility across fields. However, the community aspect became equally pivotal. The implementation of structured project management, characterized by regular meetings and systematic review processes, ensured high-quality outputs. This comprehensive approach facilitated the creation of a versatile tool adaptable across various domains, underpinned by a commitment to quality, standardization, and community engagement (4.1, 4.3.2).
2. The cultivation of open-source community around Baler extended beyond accessibility of the code. It was about nurturing a collaborative environment with a shared purpose. Key strategies included clear licensing, contributing guidelines, and inclusive community policies that welcomed diverse skill sets. Regular community meetings, and recognition of contributions fostered a sense of belonging and encouraged ongoing

engagement. This holistic methodology was crucial in creating a sustainable ecosystem around Baler, driving both user adoption and contributor retention (4.1, 4.3.1, 4.3.2).

3. To evaluate Baler's adoption and impact, a multi-faceted approach should be adopted. Key metrics include *developer activity* as seen in Figure 4.1 which gives insight into how frequently the tool is updated and refined, and *market success* which looks at the number of downloads, contributions, and mentions in industry or research. Additionally, the size and growth of its user base can provide insights into its success. Lastly, examining the regularity and quality of collaborations, code contributions, and user feedback can provide a holistic view of Baler's influence and resonance in the open-source domain (3.2.5, 4.1).

Using Baler as a case study, this thesis offers insights into the complexities and considerations required when managing open-source projects in a scientific setting. The aim has been to shed light on the nuances of moving from closed to open-source software and to provide guidance for similar future endeavors.

In the process of transitioning Baler into a well-functioning open-source software project, several significant challenges and learning opportunities emerged. The journey, demanding substantial time and resources, involved not just the technical aspects of software development but also a multidisciplinary approach leveraging expertise from the fields of particle physics and computer science.

This chapter will describe the author's observation after having spent five months monitoring and contributing to the Baler project. An overview of the key statistics representing the progress and enhancements made to the Baler project before and after the research intervention is summarized in Table 4.1. These metrics provide a quantitative measure of the project's evolution, emphasizing the effectiveness of various strategies and efforts implemented throughout this study. Figure 4.1 further explains the development rhythm of the Baler project during the period of observation. The surge in activity in between February and April 2023 reveal a period of iterative problem identification, solution design (and implementation), and evaluation – a hallmark of the design science approach which was implemented more successfully during on-site hackathons. This was when the core development of Baler was intensely underway. The subsequent decline aligns with the shift in focus to the finalization of scientific articles and the onset of the vacation periods. This graphical representation, while complementing the quantitative metrics in Table 4.1, underscores the interplay between methodological rigor and practical application throughout the project's development milestones.

4.1 Project Management Insights

Transitioning Baler was not without its hurdles. Contributors, primarily scientists entering the realm of software development, faced a steep learning curve. They grappled with unfamiliar development tools, practices, and standards, diving into new territories like VCS, testing, continuous integration, and code documentation. The choice of an open-source license, a decision with far-reaching implications, presented another challenge, with numerous nuanced options adding complexity to the process. Technical obstacles were also abundant, necessitating dedicated time for troubleshooting, research, and experimentation.

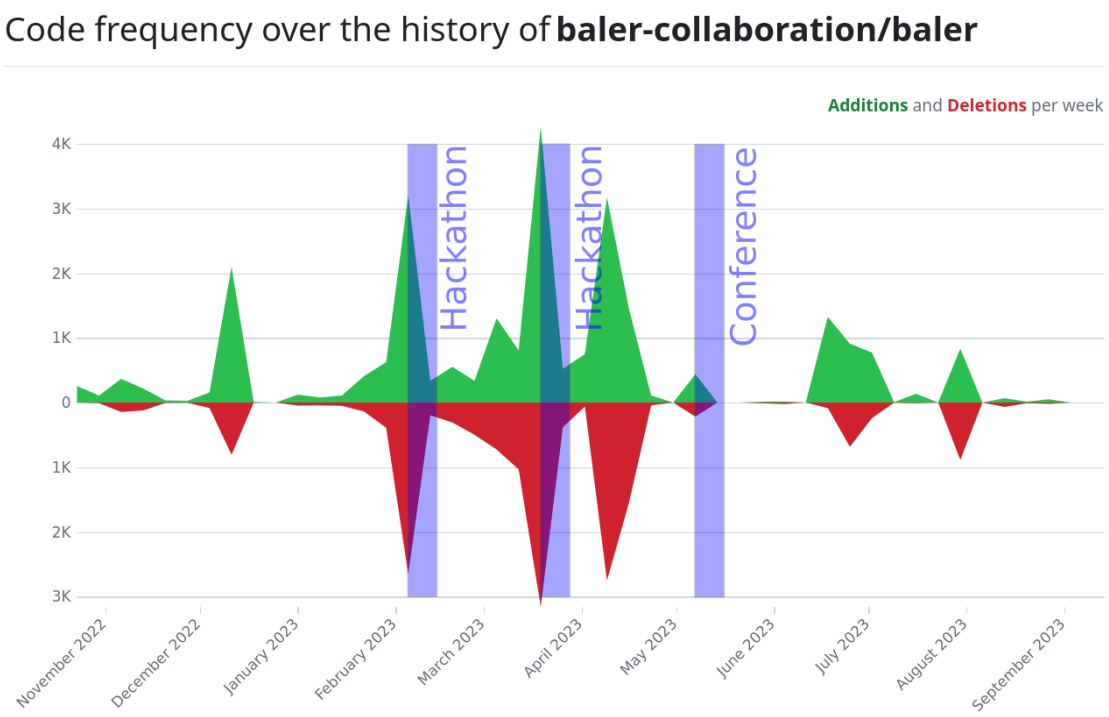

Figure 4.1: Code frequency over the history of the Baler project superimposed with major events

Despite these challenges, the team's journey was marked by significant learning and growth, often resonating with the academic spirit. Overcoming these barriers showcased their technical skill, persistence, and commitment to evolving Baler into a tool accessible to the broader scientific community.

The Baler project stands as a testament to the power of interdisciplinary collaboration. While rooted in particle physics, the project drew profoundly on computer science expertise. Computer scientists guided non-developer contributors through the maze of technical software development aspects, introducing essential practices like version control, code reviews, testing, and weekly sprint meetings. Their role was crucial in standardizing code, implementing best practices, and making pivotal decisions impacting the project's future scalability and success.

Moreover, acknowledging contributors by attributing their work, such as including their names in scientific papers, played a significant role in incentivizing participation. This form of recognition not only highlighted individual contributions but also reinforced a sense of accomplishment and belonging within the community, thereby enhancing individuals' willingness to invest their time and expertise.

The synergy between the fields was not just beneficial but necessary, ensuring the software's relevance, practicality, and usability in the scientific community. This collaboration highlighted the invaluable process of blending scientific research with robust software development practices. Despite the challenges, the project facilitated ongoing learning, fostered engagement, and confirmed the resilience and adaptability of all contributors involved.

Table 4.1: Baler project statistics before and after the project

Metric	Before	After
Code of conduct	No	Yes
Contributors	3	9
CI/CD	No	Yes
Commits	64	710
Contribution guidelines	No	Yes
Docker release	No	Yes
GitHub Stars	0	21
Issue template	No	Yes
License	No	Yes
Standardised code format	No	Yes
Tests	No	Yes
Working example	No	Yes

4.2 Developing as a non-developer

In the following section, we present four critical aspects of software development that were particularly challenging for the Baler project in its transition to open-source, given that most of its contributors were scientists and not experienced developers. These aspects include

- Documentation
- Typing and variable naming conventions
- Testing
- Version Control Systems

We explore the difficulties faced in maintaining consistency and clarity in documentation, and the obstacles encountered in promoting good practices for typing and variable naming among contributors who are new to programming. The necessity of implementing unit tests and integration tests to ensure code quality, stability, and maintainability is also emphasized, along with the challenges experienced in enforcing these practices. Lastly, we delve into the complexities of using version control systems effectively.

Through a detailed examination of these aspects, we aim to provide insights into the unique challenges faced by the Baler project and the steps taken to address them, ultimately improving the project's overall quality, efficiency, and collaboration.

4.2.1 Documentation

Documentation plays a crucial role in the success of any software project, as it ensures the project's maintainability, understandability, and usability. However, creating comprehensive and effective documentation can be challenging, particularly when working with non-developers, such as scientists without a computer science background. This complexity stems from several factors, including the need to address operating system compatibility, library

dependencies, workarounds for known issues, and potential pitfalls while following some guide.

Moreover, maintaining up-to-date documentation is essential, especially in projects like Baler that involve a continuous rotation of contributors, a characteristic shared by both academia and open-source communities. In the Baler project, it became evident that expecting scientists to simultaneously learn a programming language and implement a documentation scheme was demanding. This led to disagreements and confusion regarding documentation practices, with some documentation being overly specific to particular operating systems, or containing instructions beyond the project's scope.

Throughout the Baler project, significant strides have been made to improve the quality and consistent of documentation from a computer science perspective. The project now features a standardized and platform-independent README-file, along with well-structured in-code documentation. These enhancements facilitate a smoother onboarding process for new developers, but also foster a shared language among the core team, promoting effective communication and collaboration within the project.

During our time in the Baler project, the documentation has vastly improved, looking from a computer science view. The documentation, both in terms of a good `README`-file and docstrings [67] have been standardized and made system independent. This also lets other developers more easily get up to speed with the project, and has created a communication standard for the core developers to use, in the project, and with each other.

4.2.2 Typing

Proper typing and variable naming are essential aspects of code quality and maintainability, particularly in a programming language like Python, which features dynamic typing. The absence of static typing can lead to confusion and potential errors when working with variables of different data types, as well as difficulties in understanding the intended purpose of a variable.

Figure 4.2 illustrates a fraction of the improvements made to the Baler project, by comparing the normalization function before and after refactoring. As seen in 4.2(a), the function doesn't adhere to the pep8 style guide [67], includes code that can never be reached (code after return statement), and global variables. In contrast, Figure 4.2(b) has added some typing, removed redundant code and improved the general readability of the function.

In the early stages of the Baler project, improper typing and ambiguous variable names were prevalent, which made it challenging for team members to comprehend the code and created complications during the documentation process. Code snippets similar to `my_cat = create_dog()` were not only confusing but also prone to errors during refactoring, as the variable name suggested a `Cat` type while the method returned a `Dog`. This type of code is problematic when moving to open-source because it can create barriers to entry for potential contributors. In open-source projects, contributors often come from diverse background and may not be familiar with the specific coding habits of the original developers. As a result, ambiguous or misleading naming conventions can cause confusion and slow down the process of understanding and modifying the code.

To tackle these problems, the Baler project implemented a strategy to use more descriptive and meaningful variable names, complemented by a thorough application of typing. By embracing a uniform naming convention and employing Python's type hinting features, the

```
def normalize_data(df,config):
    if  config["custom_norm"] == True:
        pass
    elif config["custom_norm"] == False:
        global min_max_scaler
        min_max_scaler = MinMaxScaler()

        df = np.transpose(np.array(df))

        scaled_df = min_max_scaler.fit_transform(df)

        global scaling_array
        scaling_array = min_max_scaler.scale_

        df = pd.DataFrame(scaled_df.T,columns=cleared_column_names)
    return df
    epoch_loss = running_loss / counter
    print(f" Train Loss: {loss:.6f}")
    return epoch_loss
```

(a) The normalize function before

```
def normalize(data, custom_norm: bool):
    data = np.array(data)
    if custom_norm:
        pass
    elif not custom_norm:
        true_min = np.min(data)
        true_max = np.max(data)
        feature_range = true_max - true_min
        data = np.array(
            [((i - true_min) / feature_range) for i in data]
        )
    return data
```

(b) The abbreviated normalize function after, docstring left out

Figure 4.2: Comparison of the normalization function before and after refactoring in the Baler project

project has enhanced its readability and maintainability. This strategy provides clarity about the intended purpose and data type of variables. Additionally, it simplifies the documentation process by presenting a clear understanding of how variables are used across the code base.

4.2.3 Testing

Upon joining the Baler project, it became apparent that there was a lack of testing infrastructure, specifically in terms of unit tests and integration tests. This absence of testing mechanisms posed several challenges for the development process, especially in an open-source environment, which are described below.

Release Process: Without proper tests in place, it was difficult to confidently create and deploy a release, as the functionality and stability of the code could not be guaranteed. This uncertainty hindered the project's progress and risked introducing errors or regressions into the deployed software.

Refactoring and Feature Implementation: The lack of tests complicated refactoring efforts and the addition of new features. Non-developer team members were often unaware that changes to the code base required corresponding modifications or additions to the tests. This oversight led to situations where the code appeared to be broken due to outdated or incomplete test, rather than actual issues in the implementation.

Code Submission Practices: It became evident that not all team members were running tests before pushing their code to the remote repository. When one researcher was asked if his new feature had passed the tests, he confidently replied, "Yes, it did.". Only later did the team realize that he hadn't written a test for his new feature, which inadvertently broke other parts of the code. Such incidents underscored the gaps in testing awareness among contributors. This led to the potential introduction of problematic code into the project, compounding the challenges posed by the absence of testing infrastructure. To mitigate these challenges and improve the overall quality and maintainability of the Baler project, several steps were taken.

Establishing a Testing Infrastructure: The project introduced a comprehensive set of unit tests and integration tests to validate the functionality and compatibility of individual components and their interactions. This testing infrastructure provided a safety net for developers, ensuring that their changes did not inadvertently break existing functionality.

Educating Non-Developer Team Members: To bridge the knowledge gap, non-developer team members were educated about the importance of tests and their role in the development process. This training helped them understand the need to update or create tests when refactoring code or adding features, reducing instances of apparent code breakage due to outdated or incomplete tests.

Implementing Code Submission Guidelines: The team adopted a set of best practices for code submission, including forking and the requirement to run tests before submitting a pull request to the remote repository. These guidelines ensured that all team members followed a consistent process, minimizing the risk of introducing issues into the code base. Through these efforts, the Baler project successfully addressed the challenges posed by the initial lack of testing infrastructure and established a robust and reliable testing framework to support ongoing development collaboration.

4.2.4 Formatting

Just as there were a lack of tests, formatting guidelines were also missing when joining the Baler project. The lack of consistency in code formatting led to problems for the development process.

Version Control Noise: The absence of a consistent formatting style resulted in developers inadvertently introducing their personal code styles when making changes. This inconsistency generated noise in the version control system, suggesting extensive modifications when, in reality, only minor alterations had been made.

Code Review Difficulties: The lack of standardized formatting made it challenging to review code changes accurately, as it was unclear what had been genuinely modified and what was simply a formatting difference. This confusion slowed down the review process and increased the likelihood of overlooking important changes.

Enforcement Challenges: The introduction of standardized formatting guidelines presented an additional challenge, as it was difficult to ensure that all contributors adhered to the established style. This issue was particularly relevant for non-developer team members who were new to coding and more focused on functionality than formatting.

To handle these challenges, the Baler project did several things. The project adopted a standardized formatting style to ensure consistency across the code base. This unified style eliminated noise in the version control system, facilitated more accurate code reviews, and contributed to the overall readability and maintainability of the code.

Team members were educated on the importance of adhering to standardized formatting guidelines. This training helped them understand the impact of formatting on the development process and encouraged them to prioritize consistent formatting alongside functional code.

Lastly, formatting enforcement was introduced. To ensure adherence to the established formatting style, the team introduced automated tools and pre-commit hooks that checked and enforced the formatting guidelines. This approach minimized the reliance on manual enforcement and made it easier for contributors to maintain consistent formatting across their submissions.

The Baler project successfully addressed the challenges posed by the initial lack of standardized formatting and established a consistent and maintainable code base.

4.2.5 Version Control Systems

In the Baler project, employing Version Control Systems (VCS) effectively presented a significant challenge, particularly for scientists who were new to the industry standard of software development and unfamiliar with the intricacies of VCS. One of the most pressing issues was the submission of large commits and pull requests (PRs), which made it difficult to review and manage changes. The lack of rebasing to the main branch before creating PRs further compounded the problem, resulting in broken code and increased time spent on VCS management.

"I feel like I've spent more time trying to fix broken merges than I have writing new code...", a researcher laughingly exclaimed during a project meeting. The sentiment underscored the time and effort required to learn proper handling of VCS.

Navigating the complexities of VCS often demanded more time and effort than writing

the code itself, with tasks such as merging branches, deciding which branch to branch from, and handling merge conflicts proving to be particularly challenging. Moreover, the unintuitive nature of the Git command-line interface added to the confusion experienced by team members who came from non computer science backgrounds.

To overcome this, learning to use Git effectively became an essential skill for many contributors in the Baler project. Emphasizing the importance of smaller, more manageable changes, team member were encouraged to submit commits that were only a few lines long rather than a few hundred lines. This approach facilitated easier code reviews and reduced the likelihood of errors or conflicts.

Another aspect of effective VCS usage was writing clear and informative commit messages. By crafting meaningful descriptions of their changes, contributors enabled other team member to quickly understand the purpose and scope of each commit, streamlining collaboration and ensuring a better development process.

Through continuous learning and the adoption of best practices, the Baler project team improved their use of VCS, fostering better collaboration and enhancing overall project management. By focusing on smaller, more manageable commits, providing informative commit messages, and effectively utilizing Git features such as rebasing, the team successfully addressed the challenges posed by VCS.

4.3 Project Dilemmas

4.3.1 License Choice

Selecting an appropriate license for the Baler project was a challenge. The license under which an open-source project is released is a crucial decision as it sets the terms for use, modification, and distribution of the software. It was initially assumed that picking an appropriate license would be a straightforward process. However, the reality proved to be quite the opposite. The process was complicated and time-consuming, often leading to more questions than answers.

Several discussions were held within the team, and with various individuals knowledgeable about software licensing. Additionally, extensive research was conducted to understand the potential implications of each license type. However, despite these efforts, finding clear, concise, and accurate information on how to choose a license remained a difficult task. The available resources were often complex, legalistic, and contradictory, making it hard to compare and understand the potential implications of different licenses.

Amidst the discussions and challenges, one of the researchers expressed their frustration, remarking, "How can we confidently decide on a license when even the organization we're conducting research for seems unsure?". This sentiment highlighted the inherent complexities and lack of clarity surrounding the decision-making process for open-source licensing.

Interestingly, it was noted that some open-source projects had chosen to use the `CCBY4` license [68]. This license, typically used for creative works, is generally not recommended for software due to its ambiguity concerning source code and executable programs. This example further illustrates the complexity of open-source licensing and underscores the need for more accessible and clear resources to guide individuals trying to choose an appropriate license for their open-source project.

After thorough research and several discussions, the team decided on the Apache 2.0 License [59] for the Baler project [7]. This license was chosen due to its permissive nature, and widespread recognition within the open-source community. It offers a good balance, encouraging both the use and modification of the software, while also providing a legal framework to protect the project and its contributors.

4.3.2 Choice of Data Format

As we aimed to shape Baler into a tool that would support data formats from a variety of scientific fields and industries, deciding on an appropriate data format was important. Initially, we considered allowing each user to implement their own data parsing methods, potentially enabling Baler to support a variety of data formats. During one of our discussions, one research optimistically noted, "If we allow everyone to contribute their own parsers, we will end up with a ton of supported data formats!". Yet, while the idea seemed promising, it became clear that managing and updating a multitude of user-contributed parsers would be too complex.

Given these factors, we decided to go with the `NumPy`[34] data format. Why?

Firstly, `NumPy` arrays are already used across a variety of scientific disciplines. By selecting this commonly adopted format, we aimed to remove any unnecessary hurdles for potential users. There would be no need for user to convert their data into an unfamiliar or less commonly used format before they could use Baler. This consideration was particularly important, given our goal of ensuring that Baler could be used by as a large group of users as possible.

Furthermore, our choice was also influenced by technical considerations. Baler uses `PyTorch` [33], a library that leverages tensors for computations, as its core. Since `NumPy` arrays and `PyTorch` tensors share structural and functional similarities, the conversion process between these two formats is straightforward and efficient. Choosing the `NumPy` format aligns seamlessly with the technical requirements of Baler's core library.

While choosing the `NumPy` data format may limit Baler's immediate compatibility with other data formats, the advantages offered by the `NumPy` format led us to believe it was the optimal choice. We anticipate that users with different data formats can easily convert their data into `NumPy` arrays, thereby making Baler a universally applicable tool.

4.4 Reflection on Design Science

Revisiting the methodology chapter 2.3, the Design Science approach was detailed as a guiding paradigm for this research. It is worth noting how this methodology has been applied in practice within the context of the Baler project results.

4.4.1 Real-world Problem Solving

The Baler project, at its core, was a manifestation of the real-world problem-solving paradigm central to Design Science. The challenges it faced, predominantly arising from the unique composition of its contributors, serve as the major identified problems, intricately aligned with Design Science's problem identification phase.

4.4.2 Iterative Design and Evaluation

As the results unravel, the iterative nature of addressing each software development challenge – from documentation to version control – mirrors the solution design and evaluation phases in the Design Science methodology. The solutions developed were tailored to cater to the distinctive nature of the Baler project, ensuring relevance and feasibility. These solutions underwent cycles of implementation, evaluation, and refinement, echoing the iterative advancement central to Design Science.

4.4.3 Documented Learnings as Technological Rules

The learnings derived from the Baler project, as detailed in the results chapter, can be perceived as "technological rules" in the realm of Design Science. These rules or principles, formed based on real-world observation and experiences, are generalizable and can guide similar projects that involve non-developers in the software development process. The improvements made to the Baler project, particularly in documentation and code quality (typing and formatting), serve as tangible guidelines for future endeavors.

In essence, the Baler project results exemplify the Design Science approach in action. The challenges, solutions, and derived principles align seamlessly with the phases and ethos of the Design Science methodology, reinforcing its utility and relevance in practical software engineering contexts.

Chapter 5
Open-source Checklist

Based on the readings, observations, and contributions made during the period of this thesis project, this chapter provides a check-list of some key components that researchers should pay extra attention to if they want to transition their software into open-source projects. This check-list should serve as a guide on topics which should be implemented in order to have a better chance to create a good open-source project. This is a non-exhaustive list, but based on the findings of this thesis, the items in this list are the most impactful whilst also considering the effort needed for their implementation.

Before making a research software open-source project, make sure that you have the rights to do so in regards to intellectual property and copyright.

5.1 Higher Priority

- ☐ **Choosing and correctly applying an appropriate license**: Select a license that reflects your values and how you want your project to be used, ensuring it is compatible with other projects with which it might interact. This decision can influence the project's adoption, contribution, and future development.

- ☐ **Crafting a well-structured `README`**: This file is often the first item viewed by visitors to your repository. It should clearly explain the purpose of the project, its installation and usage, and how to contribute, thereby setting the tone for the entire project.

- ☐ **Providing working examples along with relevant data**: Demonstrations of your project in action, using real or sample data, help users understand its capabilities and how they might use it. This tangible insight can be crucial for engaging potential users or contributors.

- ☐ **Maintaining proper dependency control**: Clearly specify and manage dependencies to prevent conflicts, ensure consistent environments for all contributors, and simplify

the installation process for users.

- ☐ **Implementing a testing framework**: A robust suite of tests helps validate code functionality, making it easier to identify and fix bugs, thereby enhancing code quality and reliability.
- ☐ **Choose a code formatter**: This ensures consistency across the codebase, making it more readable and maintainable. It also standardizes the style, leading to fewer debates about code aesthetics.
- ☐ **Issue tracker and Kanban board for project management**: These tools help manage tasks, track bugs, and visualize the workflow, fostering better organization and prioritization within the team.
- ☐ **Implementing code reviews**: Reviews improve code quality and foster learning and collaboration among team members. They ensure that merges into main branches are well-validated, maintaining the health of the codebase.
- ☐ **Provide a quick course in version control to core developers**: Knowledge of version control is essential as it helps manage changes, tracks the history of the project, and fosters effective collaboration.

5.2 Lower Priority

- ☐ **Implementing descriptive and comprehensive typing**: Adding detailed type descriptions helps maintain code quality, makes the codebase more readable, and aids in debugging.
- ☐ **Equipping the team with a good Integrated Development Environment (IDE):** A capable IDE can enhance developer productivity through advanced code editing, debugging, navigation features, and integrated tooling.
- ☐ **CI/CD tool for continuous integration and deployment**: These practices facilitate regular code integration and automated testing, ensuring that the codebase remains clean and dependable. They also streamline the release process.
- ☐ **Creating clear contributing guidelines and code of conduct**: Setting expectations for behavior and contributions helps create a welcoming environment, encouraging participation from a diverse group of contributors.

Chapter 6
Conclusion

In concluding this thesis, we see that Baler's move to open-source is reflective of a growing trend towards collective problem-solving in technology. The insights from this project shed light on the nuanced journey that comes with adopting open-source models. It's a path that involves not just technical adjustments but also the fostering of a community where everyone can contribute meaningfully. Baler's story is one of many that show how sharing resources and expertise can lead to more efficient and innovative outcomes.

Additionally, the steps taken by Baler to become open-source resonate with wider movements in academia and the public sector, where sharing knowledge is increasingly seen as crucial for development and innovation. The practices and principles we've examined here offer a framework that can be adapted by others, highlighting the value of open communication, community engagement, and cross-disciplinary collaboration.

Open-source projects like Baler are becoming vital in bridging the gap between technology and its users, ensuring that tools are not just available but are evolving through collective input. This thesis serves as a narrative of Baler's transformation and as a resource for those considering a similar path, providing a clearer understanding of what it takes to nurture an open-source project.

In the academic sphere, open-source software like Baler presents a compelling case for a more open and collaborative approach to research and development. The transition of Baler into the open-source domain emphasizes the critical role that academic institutions play in nurturing innovation. By embracing open-source practices, academia can accelerate the exchange of ideas and tools, democratizing knowledge and fostering a culture of shared learning. This thesis has highlighted how open-source methodologies not only benefit the individual project but also contribute to a larger ecosystem of research, supporting a more interconnected and resourceful academic community.

As this study wraps up, the story of Baler is just one example within the ongoing evolution of open-source projects. It illustrates the broader potential for collaborative innovation in our increasingly digital world. The practical knowledge gained here is an invitation for others to join the open-source movement, leveraging shared knowledge for mutual benefit

and progress.

The journey of Baler, documented through this research, underscores the importance of community and collaboration in the growth of open-source initiatives. It suggests a future where shared development leads to tools that are more adaptive and accessible, and where the spirit of open-source continues to encourage widespread participation and innovation.

References

[1] CERN. CERN. https://home.cern, 2023. Accessed: 2023-05-08.

[2] Georges Aad, Tatevik Abajyan, B Abbott, J Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, 2012.

[3] Nabila Aghanim, Yashar Akrami, Mark Ashdown, J Aumont, C Baccigalupi, M Ballardini, AJ Banday, RB Barreiro, N Bartolo, S Basak, et al. Planck 2018 results-VI. Cosmological parameters. *Astronomy & Astrophysics*, 641:A6, 2020.

[4] CERN. The Large Hadron Collider. https://home.cern/science/accelerators/large-hadron-collider. Accessed: 2023-02-04.

[5] Barisits, Martin, Borodin, Mikhail, Di Girolamo, Alessandro, Elmsheuser, Johannes, Golubkov, Dmitry, Klimentov, Alexei, Lassnig, Mario, Maeno, Tadashi, Walker, Rodney, and Zhao, Xin. ATLAS Data Carousel. *EPJ Web Conf.*, 245:04035, 2020.

[6] Benedikt Hegner Graeme A Stewart. Time to adapt for big data. https://cerncourier.com/a/time-to-adapt-for-big-data, 2018. Accessed: 2023-03-27.

[7] Per Alexander Ekman, Axel Gallén, Pratik Jawahar, Fritjof Bengtsson, Oliver Woolland, Caterina Doglioni, Marta Camps Santasmasas, Nicola Skidmore, and Alma Orucevic-Alagic. baler-collaboration/baler: v1.0.0. https://doi.org/10.5281/zenodo.7817467, April 2023.

[8] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.

[9] European Open Science Cloud. European Open Science Cloud. https://eosc-portal.eu, 2023-01-01. Accessed: 2023-05-22.

[10] Fritjof Bengtsson, Caterina Doglioni, Per Alexander Ekman, Axel Gallén, Pratik Jawahar, Alma Orucevic-Alagic, Marta Camps Santasmasas, Nicola Skidmore, and Oliver Woolland. Baler–Machine Learning Based Compression of Scientific Data. *arXiv preprint arXiv:2305.02283*, 2023.

[11] Gallén, Axel. An Open-Source Autoencoder Compression Tool for High Energy Physics, 2023. Lund Student Paper.

[12] Wulff, Eric. Deep Autoencoders for Compression in High Energy Physics. 2020. Lund Student Paper.

[13] Baler-compressor. Baler. `https://github.com/baler-compressor/baler`, 2022. Accessed: 2022-12-19.

[14] Per Runeson and Martin Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical software engineering*, 14:131–164, 2009.

[15] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, et al. Systematic Literature Reviews. *Experimentation in software engineering*, pages 45–54, 2012.

[16] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How Software Engineering Research Aligns with Design Science: A Review. *Empirical Software Engineering*, 25:2630–2660, 2020.

[17] Gali Halevi, Henk Moed, and Judit Bar-Ilan. Suitability of Google Scholar as a source of scientific information and as a source of data for scientific evaluation—Review of the Literature. *Journal of Informetrics*, 11(3):823–834, 2017.

[18] Judy F Burnham. Scopus Database: A Review. *Biomedical digital libraries*, 3(1):1–8, 2006.

[19] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[20] The Institute of Electrical and Electronics Engineers. About IEEE. `https://www.ieee.org/about/index.html`. Accessed: 2023-09-03.

[21] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*, volume 9783642290442. Springer-Verlag Berlin Heidelberg, 2012. Cited by: 2912; All Open Access, Green Open Access.

[22] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley and Sons, 2012. Cited by: 944.

[23] Richard Heeks. Information systems and developing countries: Failure, success, and local improvisations. *Information Society*, 18(2):101 – 112, 2002. Cited by: 834; All Open Access, Green Open Access.

[24] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. page 499 – 508, 2007. Cited by: 423.

[25] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical Research Methods in Software Engineering. *Empirical methods and studies in software engineering: Experiences from ESERNET*, pages 7–23, 2003.

[26] Robert K Yin. *Case study research: Design and methods*, volume 5. sage, 2009.

[27] The Python Software Foundation. Python. `https://www.python.org`, 2023. Accessed:2023-04-11.

[28] Docker Inc. Docker. `https://www.docker.com`, 2023. Accessed:2023-05-03.

[29] Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. Autoencoder and its various variants. In *2018 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 415–419. IEEE, 2018.

[30] Chunyong Yin, Sun Zhang, Jin Wang, and Neal N Xiong. Anomaly detection based on convolutional recurrent autoencoder for IoT time series. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(1):112–122, 2020.

[31] Tong Liu, Jinzhen Wang, Qing Liu, Shakeel Alibhai, Tao Lu, and Xubin He. High-Ratio Lossy Compression: Exploring the Autoencoder to Compress Scientific Data. *IEEE Transactions on Big Data*, 9(1):22–36, 2023.

[32] KR Srinath. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, 4(12):354–357, 2017.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*, 32, 2019.

[34] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[35] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux journal*, 2014(239):2, 2014.

[36] Sachchidanand Singh and Nirmala Singh. Containers & Docker: Emerging roles & future of Cloud technology. In *2016 2nd international conference on applied and theoretical computing and communication technology (iCATccT)*, pages 804–807. IEEE, 2016.

[37] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[38] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[39] Hong Zhu and Ian Bayley. If docker is the answer, what is the question? In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 152–163. IEEE, 2018.

[40] Jürgen Cito and Harald C Gall. Using Docker containers to improve reproducibility in software engineering research. In *Proceedings of the 38th international conference on software engineering companion*, pages 906–907, 2016.

[41] Mozilla Foundation. History of the Mozilla Project. `https://www.mozilla.org/en-US/about/history/`. Accessed: 2023-06-12.

[42] Del Nagy, Areej M. Yassin, and Anol Bhattacherjee. Organizational Adoption of Open Source Software: Barriers and Remedies. *Commun. ACM*, 53(3):148–151, mar 2010.

[43] Linus Torvalds. Linux kernel. `https://github.com/torvalds/linux`, 2023. Accessed:2023-02-13.

[44] William Stallings. The Windows Operating System. *Operating Systems: Internals and Design Principles*, 2005.

[45] Nicholas Economides and Evangelos Katsamakas. Linux vs. Windows: A comparison of application and platform innovation incentives for open source and proprietary software platforms. In *The Economics of Open Source Software Development*, pages 207–218. Elsevier, 2006.

[46] Qiang Tu et al. Evolution in open source software: A case study. In *Proceedings 2000 International Conference on Software Maintenance*, pages 131–142. IEEE, 2000.

[47] Neeshal Munga, Thomas Fogwill, and Quentin Williams. The adoption of open source software in business models: a Red Hat and IBM case study. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 112–121, 2009.

[48] Christof Ebert. Open Source Drives Innovation. *IEEE Software*, 24(3):105–109, 2007.

[49] S. Raghunathan, A. Prasad, B.K. Mishra, and Hsihui Chang. Open source versus closed source: software quality in monopoly and competitive markets. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 35(6):903–918, 2005.

[50] Margaret S. Elliott and Walt Scacchi. Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration. In *Proceedings of the 2003 ACM International Conference on Supporting Group Work*, GROUP '03, page 21–30, New York, NY, USA, 2003. Association for Computing Machinery.

[51] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern Code Review: A Case Study at Google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, pages 181–190, 2018.

[52] Brian Witten, Carl Landwehr, and Michael Caloyannides. Does open source improve system security? *IEEE Software*, 18(5):57–61, 2001.

[53] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Communications of the ACM*, 50(1):79–83, 2007.

[54] Jing Wang, Patrick C Shih, and John M Carroll. Revisiting Linus's law: Benefits and challenges of open source software peer review. *International Journal of Human-Computer Studies*, 77:52–65, 2015.

[55] Open Source Initiative. Open Source Initiative. `https://opensource.org`, 2023-01-01. Accessed: 2023-10-14.

[56] Free Software Foundation. Free Software Foundation. `https://www.fsf.org`, 2023-01-01. Accessed: 2023-10-14.

[57] Open Source Initiative. Open Source Definition. `https://opensource.org/osd/`, 2023-01-01. Accessed: 2023-10-14.

[58] GNU. The GNU General Public License v3.0. `https://www.gnu.org/licenses/gpl-3.0.en.html`, 2007. Accessed: 2023-02-05.

[59] Apache. Apache License. `https://www.apache.org/licenses/LICENSE-2.0`, 2004. Accessed: 2023-02-05.

[60] Jerome H Saltzer. The origin of the "MIT license". *IEEE Annals of the History of Computing*, 42(4):94–98, 2020.

[61] GNU. GNU Lesser General Public License. `https://www.gnu.org/licenses/lgpl-3.0.en.html`, 2007. Accessed: 2023-02-05.

[62] Arnoud Engelfriet. Choosing an Open Source License. *IEEE Software*, 27(1):48–49, 2010.

[63] Bahar Gezici, Nurseda Özdemir, Nebi Yılmaz, Evren Coşkun, Ayça Tarhan, and Oumout Chouseinoglou. Quality and Success in Open Source Software: A Systematic Mapping. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 363–370. IEEE, 2019.

[64] Vishal Midha and Prashant Palvia. Factors affecting the success of Open Source Software. *Journal of Systems and Software*, 85(4):895–905, 2012.

[65] J. Linåker, H. Munir, K. Wnuk, and C.E. Mols. Motivating the contributions: An open innovation perspective on what to share as open source software. *Journal of Systems and Software*, 135:17–36, 2018.

[66] Johan Linåker, Gregorio Robles, Deborah Bryant, and Sachiko Muto. Open source software in the public sector: 25 years and still in its infancy. *IEEE Software*, 40(4):39–44, 2023.

[67] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style Guide for Python Code. PEP 8, 2001.

[68] Creative Commons. Creative Commons Attribution 4.0 International Public License. `https://creativecommons.org/licenses/by/4.0/legalcode`, 2013. Accessed: 2023-05-24.

EXAMENSARBETE Facilitating the Development and Release of Research Software into an Open-Source Project
STUDENT Fritjof Bengtsson
HANDLEDARE Alexander Ekman, Alma Orucevic-Alagic, Caterina Doglioni
EXAMINATOR Martin Höst

Baler – Hur öppen källkod kan tjäna ett större syfte

POPULÄRVETENSKAPLIG SAMMANFATTNING **Fritjof Bengtsson**

Tänk om vetenskapliga framsteg inte var begränsade till forskare i laboratoriet, utan för vem som helst; oavsett om hen vill bidra till forskning eller experimentera på eget håll. Det har varit målet när Baler publicerats som öppen källkod.

Initialt utvecklades Baler som en resurs för partikelfysikaliska experiment. Specifikt för att komprimera den data som måste sparas för att i framtiden kunna utvärdera experimenten i sig. Baler använder sig av destruktiv komprimering med hjälp av maskininlärning. Icke-destruktiv komprimering innebär att det går att återskapa all data precis som den var innan komprimering. Tänk dig att du har skrivit en text du vill skicka till en vän. Du lägger texten i ett kuvert och skickar iväg den – ingen information går förlorad. Om du däremot hade använt dig av destruktiv komprimering, så hade du först skrivit om texten utan adjektiv och konjunktioner. På detta sätt får all text plats på ett mindre papper och ett mindre kuvert kan användas. När din vän läser denna text kan innebörden fortfarande förstås, även om grammatiken är fel. Slutligen används maskininlärning för att på bästa sätt välja ut vilka ord som ska utelämnas, så att kuvertet kan bli så litet som möjligt samtidigt som textens innebörd bevaras så gott som möjligt. Forskarna som utvecklade Baler insåg att möjligheten att komprimera data inte bara är viktig inom partikelfysik, utan inom många olika forskningsområden. På grund av det ville de göra Baler tillgänglig till forskare inom andra discipliner, liksom den bredare allmänheten. Med detta hoppades man kunna bygga ett projekt som sträckte sig över vetenskapliga discipliner och på så sätt främja samarbete och innovation. Men att göra källkod och programvara framtaget för ett specifikt ändamål allmänt tillgängligt var mer utmanande än förutspått. Om utomstående individer skulle vilja använda Baler, och eventuellt bidra till dess utveckling, behövdes en övergripande strategi för hur detta skulle uppnås. Den strategi som användes i övergången till öppen källkod baseras delvis på nuvarande forskning liksom praktisk erfarenhet från projektet. Vidare så togs en rad riktlinjer fram som ska kunna användas som grund då ett projekt skall omvandlas och publiceras som öppen källkod. Vår förhoppning är att Balers transformation från specifik programvara till ett etablerat öppet källkodsprojekt kan exemplifiera ett skifte inom vetenskapen, där nyttig programvara på ett lätt sätt kan delas mellan olika discipliner, och allmänheten. Med detta skifte skulle vi få mer valuta för de resurser som läggs på forskning, då vi underlättar disciplinövergripande samarbete och bjuder in de som i vanliga fall kanske inte hade tagit del av forskningens framsteg. Vem vet vilka insikter som väntar när vi utnyttjar världens kollektiva intelligens.