Research Software
- Publication and Sustainability

Master’s Thesis

Claudia Eitzen

November 18, 2020

KIEL UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SOFTWARE ENGINEERING GROUP

Advised by: Prof. Dr. Wilhelm Hasselbring
Dr. Reiner Jung
Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 18. November 2020
Abstract

Good scientific practice is essential to any research process, and the creation of research software (RS) has to meet this requirement. Its core standard includes intrinsic quality, transparency, reproducibility, re-usability, comparability, and transferability. Central element to promote the implementation of this standard in the research routine, is the publication of RS according to the Open Science and to FAIR Guiding Principles. RS is accessibly maintained in server operated repositories that contribute to data sustainability. The present thesis is supported by the conviction that computer related science has become a research discipline for its own. Therefore, a large-scale study of 200 thousand GitHub repositories, and of 30 thousand ACM (Association for Computing Machinery) publications and arXiv publications is realized.

The intention was to find the current sustainability state of research software in both computational science and computer science. For that purpose, the frequency of links is determined in publications to RS and of links within RS regarding computational science as well as computer science. The analyses reveal that publications referencing GitHub repositories are mainly (68%) assigned to computer science. In contrast, GitHub repositories mostly (78%) refer to publications of computational science. GitHub repositories referenced by ACM publications are the most durable ones. The remaining repositories have noticeably shorter lifespans, if they do not even consist of the single first commit. The present investigation complies with the strict demand for reproducibility as a key element of good scientific practice. This study reproduces successfully the general tendencies obtained from a pilot project in the Department of Computer Science at Kiel University in cooperation with the University of Southampton.
# Contents

1 Introduction  
1.1 Motivation .................................................. 1  
1.2 Goals ......................................................... 7  
1.3 Document Structure .......................................... 7  

2 Foundations and Technologies  
2.1 Computational Science .......................................... 9  
2.2 Research Software ........................................... 11  
2.3 Good Scientific Practice ....................................... 12  

2.3.1 Taxonomy for Reproducibility Concepts ...................... 14  
2.3.2 Research Software Engineering ............................ 15  
2.3.3 FAIR Guiding Principles ................................... 18  
2.3.4 Open and FAIR Research Software ......................... 21  
2.3.5 Observatories in Research ................................. 23  
2.4 Technologies .................................................. 24  

3 Analysis of Research Software  
3.1 Research Questions ........................................... 25  
3.2 Requirements and Constraints ................................ 26  

3.2.1 Functional Requirements ................................... 27  
3.2.2 Non-Functional Requirements ............................... 29  
3.3 Framework Architecture ....................................... 30  

3.3.1 Configuration File ........................................ 30  
3.3.2 Data Model ................................................ 32  
3.3.3 Database Module ......................................... 34  
3.3.4 Auxiliary Functions ..................................... 35  
3.3.5 Harvester Modules ....................................... 36  
3.3.6 Jupyter Notebooks ....................................... 38  
3.4 Publication Harvester ......................................... 39  

3.4.1 Search Term .............................................. 40  
3.4.2 Design .................................................... 40  
3.4.3 Implementation .......................................... 41  
3.5 Repository Harvester ......................................... 45  

3.5.1 Search Term .............................................. 45  
3.5.2 Design .................................................... 46  
3.5.3 Implementation .......................................... 49
Chapter 1

Introduction

1.1 Motivation

The development of software has become established in many areas of research and is closely linked to its strength and growth. According to the concept of Smith et al. [2016, p. 2], research is regarded as “work intended to increase human knowledge and benefit society, in science, engineering, humanities, and other areas”. In economics, political science, biology, and astrophysics, research software is applied to gain new scientific discoveries and deeper knowledge. Due to the “strong and growing dependence of research on software” [Förstner 2017], it is important that research software receives its adequate recognition “as first-order element of research” [Cohen, Katz, Barker, Chue Hong, Haines, and Jay 2020, p. 4]. In return, it must meet the requirements of good scientific practice [Förstner 2017], for instance, the demand for reproducibility, confirmability, transparency, quality, and re-usability [Steuerungsgremium der Schwerpunktinitiative „Digitale Information“ der Allianz der Deutschen Wissenschaftsorganisationen 2017]. By contrast, the current state of software development bears considerable deficiencies.

Portegies Zwart [2018] draws the attention to another serious issue, the decision against the dissemination of developed code. This may be based on different considerations. Some scientists fear the loss of their technological advantage or the refutation of their

1. Introduction

research results. An important reason is the lack of recognition for code development. To overcome this deficit, a system for software citation or source code communication should be established [Hasselbring, Carr, Hettrick, Packer, and Tiropanis 2020a]. Portegies Zwart [2018] demands a reorganization of the scientific software development process by reimplementing libraries of fundamental solvers, defining code-exchange and data-exchange strategies, and designing method-coupling paradigms. He emphasizes his claim by a comparison of a modular astrophysical system with the LEGO bricks. Here, endless permutations are enabled by an abstract fine-grained structure and a general interface. The fine-grained structure relating to software does not mean the partition into fundamental operations, but collections of fundamental elements. An advancement are the DUPLO blocks with a coarse grained structure, that represent self-contained objects.

These growing claims for reusable software and reproducible research are not restricted to astrophysics, but occur more and more frequently in many other research areas. The promotion of both objectives requires new strategies in various aspects of the research process. One core element is the publishing of scientific contributions. However, the classical system for research publication management, dominated by a few big editors [Gomez-Diaz and Recio 2020], is considered to be a 17th-century technology. In that century the roots of scientific journals can be found. Scientific cooperation had become essential, not least because of the growing research effort [Masuzzo and Martens 2017]. In the 15th and 16th century a few European polymaths focused on research. The first communication channel for research findings and claims of inventorship was the dispatch of anagrams to familiar scientists [Bartling and Friesike 2014]. The number of scientists increased over time: from over one million scientists (1850) to approximately 100 million these days. The polymath evolves into a highly specialized scientist, working in one of the various, clearly delimited academic disciplines.

In order to meet growing demands, scientific societies were founded and scientific journals produced to disseminate knowledge [Masuzzo and Martens 2017]. The French „Journal des sçavans“ and the English „Philosophical Transactions of the Royal Society“ started in 1665 and were the first journals merely for scientific content [Bartling and Friesike 2014]. In the course of time, the dissemination of knowledge enlarged, but simultaneously slowed down [Masuzzo and Martens 2017]. This is caused, for instance, by the time consuming peer review process, high subscription fees for journals, a restricted information flow [Lin 2012], and by focusing on positive research findings [Bartling and Friesike 2014]. Overcoming these obstacles with the opportunities presented by digital information technologies and the Internet advances the scientific progress [Morey et al. 2016]. For example, negative or intermediate results, insufficient to fill an entire publication, could be announced in low threshold publications, like blog posts, wiki articles, or as contribution to an online discussion, as shown in Figure 1.1. Consequently, as depicted in Figure 1.2, the lost ideas, lost knowledge, and unpublished results of the current system, could be used for “faster knowledge exchange, prevention of unnecessarily repeated experiments, and a more vivid discussion” [Bartling and Friesike 2014, p. 9]. These objectives are comprised
1.1. Motivation

The Open Science movement emerged as counterculture to the current scholarly publishing system and to counteract its closed publication practice [Masuzzo and Martens 2017] and current problems of science. The presence of this term steadily increases, it is of growing interest, and is itself an object of research, but does not yet have a clear and generally accepted definition or concept [Tennant 2018]. For instance, Vicente-Saez and Martinez-Fuentes [2018, p. 1] define Open Science as “transparent and accessible knowledge that is shared and developed through collaborative networks”. Or Open Science is considered as “the idea that scientific knowledge of all kinds should be openly shared as early as is practical in the discovery process”. Furthermore, often it is used interchangeably with other umbrella terms [Bartling and Friesike 2014] and surrounded by considerable debates, that merely have in common the need for opening science. This vaguely formulated notion causes quite variable views and a variety of concepts for prospective knowledge creation and dissemination. However, Fecher and Friesike [2014] proposed a classification

---

Figure 1.1. Knowledge dissemination: top line shows the current state in research, the lower line shows the aspired state of research [Bartling and Friesike 2014, p. 9,10]

of various umbrella terms, like Science 2.0, e-Science, Open Research, or Open Science [Bartling and Friesike 2014].

---

2http://openscience.org/an-informal-definition-of-openscience/
1. Introduction

Figure 1.2. Feasible effects of knowledge dissemination: top line shows the current state in research, the lower line shows the aspired state of research [Bartling and Friesike 2014, p. 9,10]

of the current ideas into five schools of thought. The classification comprises the infrastructure school, the public school, the measurement school, the democratic school, and the pragmatic school. They can be differentiated by means of main hypotheses, stakeholder groups, objectives, and applied methods and tools, as outlined in Figure 1.3.

Most approaches have one core aspect in common, enhancing the transparency from beginning to end of the research process [Tennant 2018]. In this context, also Open Access is a frequently mentioned notion, sometimes used interchangeably with Open Science [Lacey et al. 2020]. But Open Science must not be reduced to openly accessible research publications. This is only one of the four pillars Open Science should be based on, as stated by the EU-funded FOSTER Plus project. Furthermore, Open Science comprises the sharing of research data, an open peer review process, and open source software and workflows.3

Open Access On May 27th, 2016, the Competitiveness Council, a gathering of EU member state ministers of science, innovation, trade, and industry, decided that by 2020 all scientific

3https://www.fosteropenscience.eu/learning/what-is-open-science/#/id/5ab8ea32d1827131b98e3ac
1.1. Motivation

Open Science
Infrastructure School
Assumption: Efficient research depends on the available tools and applications.
Goal: Creating openly available platforms, tools and services for scientists.
Keywords: Collaboration platforms and tools

Public School
Assumption: Science needs to be made accessible to the public.
Goal: Making science accessible for citizens.
Keywords: Citizen Science, Science PR, Science Blogging

Democratic School
Assumption: The access to knowledge is unequally distributed.
Goal: Making knowledge freely available for everyone.
Keywords: Open access, intellectual property rights, Open data, Open code

Measurement School
Assumption: Scientific contributions today need alternative impact measurements.
Goal: Developing an alternative metric system for scientific impact.
Keywords: Altmetrics, peer review, citation, impact factors

Pragmatic School
Assumption: Knowledge-creation could be more efficient if scientists worked together.
Goal: Making the process of knowledge creation more efficient and goal oriented.
Keywords: Wisdom of the crowds, network effects, Open Data, Open Code

Figure 1.3. Open Science – built by five schools of thought [Fecher and Friesike 2014, p. 19]

articles should be freely available. This decision has not yet been implemented. But in recent years, a lot of preprint servers have been founded, like arXiv, ChemRxiv, engrXiv, bioRxiv, socArxiv, PeerJ Preprints, AuthorOne Preprints, and OSF Preprints [Barba 2019]. Many journal publishers like Elsevier, Nature Publishing Group, Springer, and Taylor & Francis, have no objection to preceding posts on preprint servers, and those will not be assessed as prior publication.

Open Peer Review The key areas of Open Science are interconnected. Thus, Open Access has triggered changes in the peer review process. The spread of preprints and progress in community-organized peer review caused the beginning of decoupling publication from the established journals. But to increase the transparency in peer review, three issues regarding incentivisation for participation, quality control (verification), and certification and reputation have to be solved [Tennant 2018]. In the prevailing peer review process,

1. Introduction

the reviewers receive seldom appropriate incentives for their invested time and energy. Furthermore, after publication the reviews are discarded in the majority of cases. In contrast, F1000Research realized an open peer review process. The published article is complemented with each submitted review, including the reviewer’s name and comments.\footnote{https://f1000research.com/faqs}

Open Source Software  The main quality of open source software is freely available source code. Though not every source code that can be found on the Internet is open source. Software, not holding a license, is protected by copyright (all rights reserved) and may not be used for anything [Barba 2019]. According to the Open Source Initiative, Open Source promotes not only the accessibility of source code, but implies also that the distribution terms comply with ten specific criteria inspired by the Debian Free Software Guidelines. These include the free redistribution and the permission for modifications and derived works.\footnote{https://opensource.org/docs/osd} The first image of a black hole, reported on April 10, 2019, is a good example for the application of open-source software. It was generated by using Matplotlib and other open-source Python components [Nowogrodzki 2019].

Open Data  Besides the freely available publications and software, the included data, as a main part of a publication, should also be accessible to everyone interested. In view of Open Science, research data, especially raw data, and the contained information are not subject to copyright protection and there exists no “ownership on data” [Kuschel 2018, p. 764]. By U.S. law such data are regarded as facts [Barba 2019], and European copyright protects only the personal intellectual creation, for instance, a specific representation of the research data as text or figure. In the case of databases, an ancillary copyright may be adjudged, provided that a substantial investment is necessary for the data alignment or representation. Excluded are the expenses for the data acquisitions [Kuschel 2018]. Essential elements are the data maintenance and the annotation with metadata [Masuzzo and Martens 2017]. In other words, improved data management is required [Barba 2019], as it is an elementary prerequisite for processes to enhance knowledge and to promote innovation, to combine new data and knowledge with existing ones, and to enable the reuse of the data for interested parties [Wilkinson et al. 2016].

For the integration of the Open Science principles in the everyday life of science, a series of ideas and approaches for technical solutions are drawn up. But no gold standard or best practice are defined so far, especially when dealing with software. The development and dissemination of good practices requires a coordinated approach, that involves various steps. For instance, a continuing education to improve software development in the area of scientific research should be established. The effort of research software development should be acknowledged as well as the contribution of the research software to scientific research. In addition, it needs to be ensured that both the software and the data, which is
generated or applied to the software, are reasonably stored [Davenport, Grant, and Jones 2020]. Prerequisite for the realization of this approach is a basic knowledge, which includes information about the current state of research software development behavior, as well as about the publication practice for research software. The pilot study of Hasselbring, Carr, Hettrick, Packer, and Tiropanis [2019, 2020b] reveals that scientific disciplines use different methods to reference research software. Also the sustainability of research software varies regarding the research area. Within the scope of this work, the pilot study should be implemented anew to verify its findings and ensure its reproducibility.

1.2 Goals

G1: Research Software Publication and Sustainability

To provide a basic understanding for further discussions on research software publishing, as well as maintainability, the current state should be reviewed. Based on the pilot study of Hasselbring et al. [2019, 2020b], a research study is designed, whose workflow consists of three main steps. Initially, research software candidates are gathered (data acquisition). These form the basis for the identification of research software (data processing), that is analyzed regarding its method of referencing and its lifetime (data analysis).

One key aspect concerning the framework design is the required reproducibility. This term relates to the definition of Patil et al. [2016], describing the achievable minimum standard for scientific findings. Chapter 2 will describe additional information about standard demands. In accordance with Kitzes et al. [2018], the workflow should be designed as a sequence of small steps and as automated as possible. Moreover, all involved elements should be labeled and documented.

G2: Set up a Research Software Observatory

To comply with good scientific practice, this study should meet the requirement of reproducibility. The basis to this end are publicly accessible data and source code. For its realization the approach of Research Software Observatories, introduced by Hasselbring et al. [2019], is applied. This concept schedules that the implemented software is “FAIR”. The acronym stands for findable, accessible, interoperable, and reusable, pursuant to the FAIR Guiding Principles, see Section 2.3.3. To meet these requirements, metadata for both data and software are provided. The software should be made available as open source software and in a runtime environment to avoid execution problems.

1.3 Document Structure

The main contribution of the present essay is the analysis of research software, more precisely scrutinizing its sustainability and demonstrating its associated research area
1. Introduction

(Chapter 5). This analysis is based on the process of the research software identification (Chapter 3) and earmarked as Open Source Software (Chapter 4). In addition, the foundations of research software and good scientific practice are described, as well as the technologies needed for the implementation (Chapter 2). Finally, this work is completed by summing up the results and giving perspectives for future work (Chapter 7).
Due to the wide range of developments in computer technology, computational and numerical techniques are more and more frequently applied to solve large and complex problems. These improvements include the increased speed of computation, expanded efficiency of memory storage devices, improved computer hardware, and enhanced algorithms and mathematical techniques. Hence, software is applied in many areas of research to forecast real world processes and to better understand them [Kanewala and Bieman 2014]. Computational models complement the lab, but cannot be taken as substitutes. Other application areas for software in research may be found in the data processing pipeline [Barba 2019]. In this function, software has successfully found its own specific proper place in the overall search for scientific knowledge. This is confirmed by the results of a survey targeted at 417 researchers, of those 91.8% use software. In that sample, 21.1% of the researchers stated that – with more effort – their research would be feasible without software. This does not apply to 68.6% of the researchers, as software is crucial to their work [Hettrick et al. 2014]. A most one-sided situation prevails in solar-physics. $99\pm0.5\%$ of 368 surveyed researchers benefit from the use of software in their research [Bobra et al. 2020]. In astrophysics, 100% of 1,142 surveyed astronomers use software in their research, and 90% of them develop wholly or partially their own software [Momcheva and Tollerud 2015]. These results indicate that the use of software is not only restricted to computer science, but can be met in various research areas. Such broad acceptance entails the emergence of computational science as a research discipline of its own.

### 2.1 Computational Science

The interdisciplinary approach considers the development of software, algorithms, and models, which try to answer research issues. Thus, computational science has become part of the scientific research. Different concepts were created specifying this research field. One approach looks at it as an interdisciplinary effort that combines "concepts and skills from the disciplines of science, computer science, and mathematics" to solve complex scientific issues.\(^1\) So, it can be seen as additional method to gain research findings, besides the observational, experimental, and theoretical methods of scientific research. The journal Nature also describes computational science as discipline, that deals with

\(^1\)https://www.shodor.org/chemviz/overview/compsci.html
A less discipline-specific understanding of computational science describes its elements as application, algorithm, and architecture. Figure 2.1 illustrates computer science requiring computing environment in an ordered manner that is represented by its architecture. The mathematical model is developed from concepts and skills which are formally condensed in an algorithm. The latter will be deployed to investigate scientific issues. Its goal-oriented operation is called application. The model is based on theory, is fed with data from experiments, and is executed by computation.

Science itself is the core component of each computational science problem. After clearly formulating a scientific problem, a suitable mathematical model (algorithm) to answer the defined research question has to be found. To solve the generated algorithm, techniques of numerical analysis can be used. Since appropriate solutions require a high computational effort, the algorithm is implemented by means of the computer science technologies and computational software tools.¹

Also Tolk [2018, p.1] describes a similar concept with insights from the modeling and simulation discipline, whereby “[c]omputational science of every domain equals applying modeling and simulation methods”. So, the computational science simulates scientific models, namely executes them on a computer. The discovery of the Higgs boson is the result of a successful computer simulation based on the predictions of particle physics. Due to the reduction to certain predicted events, the experimenters received indications where...
2.2 Research Software

Software, created or used within a scientific discovery process or gained as a result of scientific work, is designated as research software [Förstner 2017; Hasselbring et al. 2020a]. Kanewala and Bieman [2014] label software as scientific software, applied for insights and forecasts of real world processes. More precisely, software is understood as source code with complementary documentation, parameters, and workflows [Scheliga et al. 2019], that is applied to generate, process or analyze results, intended to be published. The publication form may be a book, a conference paper, a monograph, a journal, a review, or a thesis. For example, a web search, applied in a research process, is not rated as research software. The size of the source code is not a characteristic of research software. It may consist of a few lines of code to a professionally developed software package [Hettrick et al. 2014].

Carver et al. [2011] differentiated between research software and production software, that are represented in computational science and engineering. The latter one is software developed for real users, while the first one is developed aiming to publish a paper. Hitherto, it is challenging to receive academic reputation by developing research software [Barba 2019]. The only way to gain academic reputation is to publish a paper about the research software as citeable object. The developer friendly Journal of Open Source Software (JOSS, http://joss.theoj.org), an Open Source Initiative (OSI) affiliate, aims to give research software developers the opportunity to receive citation credit and to optimize their software quality with the help of the peer-review process and a committed editorial board [Barba 2019].

Despite the various advantages of research software, its wide variety of tasks brought new problems to the scientific community. Software becomes more and more complex and its estimated operating lifetime increases. Scientific simulations are expected to have various effects affecting the run-time behavior. Along with the emergence of multi-core, heterogeneous, and distributed computing systems, these factors caused the revelation of productivity inadequacies in the scientific software development process [Johanson and Hasselbring 2018, p. 3].

Until the exploitation of the clock speed limit for single-core processors in 2004, researchers developed under the assumption that the increased performance of microprocessors, due to technological enhancements, does not require substantial modifications of the source code. Then, novel approaches for the performance improvement had to be devised. After increasing the number of processor compute cores for the Central Processing Unit (CPU), multi-core CPUs are designed and subsequently complemented with multi-core external accelerator devices. The scientist had to adjust to the new conditions of parallel computing. Their straightforward programming style resulted in source code which was hard to maintain and scale up to new hardware architectures. Also the next step in...
2. Foundations and Technologies

form of massive distributed parallel machines provoked productivity problems. Due to the increased expenditure of time for the adjustment of source code to new hardware architectures, this problem is known as productivity crisis in computational science. In addition, the collaboration of multidisciplinary teams is quite time-consuming and demands considerable administrative effort, required by the interdisciplinary aspects to increase the predictive strength of computational models [Johanson and Hasselbring 2018].

Another aspect for reliable computational results is the code quality. In this respect, a lot of scientific disciplines point out a lack of programming skills. Only 8% of 1,142 astronomers, using software in their daily work, have participated in substantial software development training. The majority of the astronomers had no training (43%), or only a limited one (49%) [Momcheva and Tollerud 2015]. The “Climategate” scandal, leaked e-mail correspondence from the researchers of the Climatic Research Unit at the University of East Anglia, revealed a lack of programming skills of the researchers. Often they are incapable of gaining new programming skills to keep up with the enlarging complexity of programming tools and computers. Moreover, testing, documenting, and releasing source code is not common practice [Merati 2010]. Because of such deficiencies, the discussion about the reproducibility of computational results evolved [Johanson and Hasselbring 2018].

Reproducibility crisis is a frequently encountered problem. This methodological crisis has revealed that in many scientific disciplines most of the research findings cannot be reproduced nor replicated. The US$1.6-million Reproducibility Project: Cancer Biology attempted to replicate the results of 50 high-profile cancer papers. But due to the missing documentation of the experiments, the project stopped after inspecting the first 18 papers [Teytelman 2018]. This problem also occurs in psychology. One hundred replication studies resulted in two thirds of studies with insignificant findings. A further discrepancy exists in the average replication effect size, that was only the half of the effect sizes in the original studies [Held and Schwab 2020]. Different causes were identified that led to this irreproducible research. These include the publication bias, low statistical power, \( P \)-value hacking, and HARKing, which describes the formulation of the hypothesis after the results are known [Bishop 2019]. Another important factor is the pressure to publish results [Vitek and Kalibera 2011] that leads to an increased frequency and thus quantity in the generation of significant findings [Held and Schwab 2020]. In computer science, especially in systems research, a lack of benchmarks, a lack of experimental methodology, and a lack of understanding of statistical methods can be added to this list [Vitek and Kalibera 2011]. These serious observations require a strengthening of research reliability, that can be achieved by the compliance with best practices during the research process.

2.3 Good Scientific Practice

The German Research Society (DFG) [Deutsche Forschungsgemeinschaft 2019] specifies six general principles and eleven guidelines for good scientific practice in the research process.
2.3. Good Scientific Practice

Addressed are both researchers and institutions. In the context of research software, the following guidelines are of special interest.

**Guideline 7** The scientist works in compliance with professional standards and explains the applied quality assurance mechanisms. One core element of the quality assurance is the replicability or confirmation of the findings by other researchers. Decisive standards and established methods are prescribed by the corresponding scientific discipline. For processes, like the data processing pipeline, and “the selection and use of research software, software development and programming, and the keeping of laboratory notebooks” [Deutsche Forschungsgemeinschaft 2019, p. 13], the continuous quality assurance has to be ensured. For instance, the source code of the selected research software has to be documented and must meet the standard of citability and persistence. Besides the publication of the research findings, the details of the used software and data, gathered or reused, have to be cited or indicated respectively.

**Guideline 11** Comparability and transferability of research findings are assured by the observance of standards regarding, among other things, the research data gathering and the use of software.

**Guideline 12** The need for documentation is not only relevant for the source code of a developed software, but is required for all steps of the research process. This improves the general understanding of the research project, the clearness of the citations, and the review of the results, and thus the preconditions for the replication.

**Guideline 13** To promote transparency and reuse, all research results, the research data, principal materials, and used software are also made available. The self developed software is provided together with its source code and an appropriate license. Upon the decision of the researcher, in certain exceptional cases the publication may be omitted, like in patent applications.

So, the core components of good scientific practice can be summarized as quality, transparency, reproducibility, re-usability [Förstner 2017], comparability, and transferability. As an essential part of the research process, research software has to meet these requirements. Various concepts are presented to promote their implementation in the daily research routine. Reproducibility and replicability are among the current central terms in the scientific community. The definitions of these associated terms and approaches are as varied as the research areas themselves. Since common grounds are missing and a common sense for best practices does not exist, scientific activities suffer from a great deal of ignorance concerning the application of reproducible research [Pontika et al. 2015].
2. Foundations and Technologies

2.3.1 Taxonomy for Reproducibility Concepts

Bechhofer et al. [2010] distinguished between reusable, repurposeable, repeatable, reproducible, and replayable as feature variants of general reuse. When describing these terms, the authors relate to Research Objects (RO), not only focusing on data. The notion RO defines a container for the information about the involved researchers, the used data, and the employed methods for the sample generation and analysis. In the case of reusability, the RO is considered as a black box and reused as a whole or single entity. If a new experiment is derived from a RO by using some of its components, like methods, a sequence of steps, or data, it is considered to be repurposeable. This requires a detailed description of the structure and the dependencies of the individual parts. Additional documentation, containing information about the execution order and the computation of the results, is also needed to repeat a study with a RO. By providing, for example, virtual machines, the repetition may be simplified. A special case of repeatability is reproducibility, which as a notion is used interchangeably with replicability. It is an important element for the validation of scientific claims. Therefore, it requires some form of comparability framework for checking whether the results are the same as in the preceding study. If the objective is defined in understanding a study design and a repetition is not necessarily essential, the replay allows the investigation of each step in the experimental procedure. Thus, Bechhofer et al. [2010] belong to the group, that does not differentiate between reproducibility and replicability, also referred to as group A [Barba 2018], see Figure 2.2. Similarly the Open Science Collaboration [2012], an open collaboration of researchers with the objective to align scientific values and scientific practices, states explicitly the synonymous use of these notions.

Within the second group, named B, there exist two different understandings of reproducibility and repeatability, as Barba [2018] pointed out. Across the scientific disciplines the terminology of group B1 is the most common one, see Figure 2.2. The minimum standard, which is defined as the application of the same methods to the same data resulting in the same findings, is designated as reproducibility. Raising the level of trust towards the gold standard, new methods are applied to new data following the original experimental setup, also referred to as replication. As an intermediate step the original methods can be used. The obtained results should correspond to the original results. The essential steps of both processes are summarized in a visual model, shown in Figure 2.3. For instance, the recommendations of the American Statistical Association (ASA) for supporting reproducible research are based on this definition.

Two large professional groups are opposed to the above described usage of the terminology, the Association of Computing Machinery (ACM) and the Federation of American Societies for Experimental Biology (FASEB). This fact significantly complicates the realization of a general consensus for all research areas. The group B2, composed of ACM and FASEB, characterizes the minimum standard as replication. In the policies about

---

3https://osf.io/vmrgu/
2.3. Good Scientific Practice

<table>
<thead>
<tr>
<th>A</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>political science</td>
<td>signal processing</td>
<td>microbiology, immunology (FASEB)</td>
</tr>
<tr>
<td>economics</td>
<td>scientific computing</td>
<td>computer science (ACM)</td>
</tr>
<tr>
<td>econometry</td>
<td>epidemiology</td>
<td></td>
</tr>
<tr>
<td>clinical studies</td>
<td>internal medicine</td>
<td></td>
</tr>
<tr>
<td>physiology (neuro)</td>
<td>computational biology</td>
<td></td>
</tr>
<tr>
<td>biomedical research</td>
<td>statistics</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2. Users of reproducibility/replicability grouped by discipline. Group A uses replicability and reproducibility synonymously. Group B1 claims reproducibility as minimum standard. Group B2 claims replicability as minimum standard [Barba 2018, p. 14].

ACM Association of Computing Machinery, FASEB Federation of American Societies for Experimental Biology

“Artifact Review and Badging” the ACM describes the notions repeatability, replicability, and reproducibility, based on the definitions of the International Vocabulary for Metrology (VIM). A study is repeated, if the same team conducts it with the same experimental setup. For a replication, a different team executes the same experimental setup. And for reproducible research, a different team uses a different experimental setup. In each case the same results are obtained.

Vitek and Kalibera [2011] added the notion repetition to the wide variety of terms. It refers to the ability to conduct the same experiment with the same methods on the same data, and hence corresponds to reproducibility respectively replicability. This should ensure the stability of the research findings and provide a baseline for the investigation of new ideas. Furthermore, repetition should reduce the likelihood of random errors on the experimenter side and simplify the review of the sufficient statistical methods for the random effects in the underlying system. Reproductions, conducted by independent research teams after the publication of a study, corroborate the stated hypothesis. This independent confirmation of findings is one core element of the scientific method and contributes to the improvement of the research reliability. As prerequisite for the reproduction of research results, the research software has to provide a minimum quality standard, that can be achieved by applying software engineering practices.

2.3.2 Research Software Engineering

The increasing awareness about research software also brought the subjects of its quality and sustainability into the focus. Both are important prerequisites for the reliability of any

---

5https://www.acm.org/publications/policies/artifact-review-badging
2. Foundations and Technologies

To address this major difficulty we need a statistical framework for the scientific process. Typically statistical and machine learning models only formally define the outputs of the scientific process with random variables. The outcome is often designated by \( Y \) and the covariates of interest by \( X \). This framework is limited because it does not allow us to model variation in what the scientists in question intended to study, how they performed the experiment, who performed the experiment, what data they collected, what analysis they intended to perform, who analyzed the data, and what analysis was actually performed. We have created a formal statistical model that includes terms for all of these components of the scientific process (see Supplemental Material). The key steps in this process can also be represented using a simple visual model (Figure 1).

Figure 1. A graphic representation of the statistical model for the scientific process

A. Reproducibility is defined as reperforming the same analysis with the same code using a different analyst; replicability is defined as reperforming the experiment and collecting new data.

B. The paper that only 6 out of 53 preclinical studies replicated only reported a question and a claim, but not the rest of the scientific components of a study.

C. The disagreement over the Reproducibility Project Psychology (9, 19) is because a replication was not performed since the population changed.

D. In the case of the controversy over genomic signatures for chemosensitivity (2), reproducibility wasn’t the main issue - the issue was that the original study did not have correct code and data.

Here we provide informal definitions for key scientific terms and provide detailed statistical definitions in the Using this modeling framework we provide formal definitions for the following terms (see Supplementary Material for formal statistical definitions and additional definitions):
2.3. Good Scientific Practice

![Research Software Engineering diagram]

*Figure 2.4. The four pillars of Research Software Engineering [Cohen et al. 2020, p. 3]*

look to the DARPA Open Catalogue. Researcher services are offered also by Nature Research, formerly known as the Nature Publishing Group. Another important factor is the consensus of a common language, since there exists a varying, sometimes conflicting terminology. The recognition area deals with the modernization of the publication process using new technologies of which F1000Research is an example. The citability of software will not at all be irrelevant. Its promotion will result in – among other things – improved career pathways for research software engineers.

These core areas can also be found in the concept of research software engineering that Cohen et al. [2020] present. Accordingly, research software engineering is based on four pillars, labeled as software development, community, training, and policy, see Figure 2.4. The community pillar is based on the need for experience exchange to learn from other developers or learn new things and acquire new knowledge. The concept of Continuing Professional Development (CPD) is not yet so strongly represented as in many industries. Therefore the initial and ongoing development of skills manifests itself in the third pillar, the training. As important as the other three is the Policy pillar, which stands for institutional and national networking to support the development of quality software. These policies

---

8 [https://www.darpa.mil/our-research](https://www.darpa.mil/our-research)
9 [https://www.nature.com/nature-research](https://www.nature.com/nature-research)
10 [https://f1000research.com/](https://f1000research.com/)
also include dedicated support for research software careers, which may be aspired to by software developing researchers as well as software practitioners.

The Software Development pillar relates to core aspects of software quality, i.e. maintainability, robustness, and sustainability. The Software Sustainability Institute, specifies sustainability as “software you use today will be available – and continue to be improved and supported – in the future”.\footnote{https://software.ac.uk/about} Due to the importance of sustainability in the process of moving towards complete automation in both research and industry, it is a relevant research field in software engineering. Though the current research is limited by the lack of practices at the software design level and sustainability benchmarks. Imran and Kosar [2019] also revealed that a lot of software is inadequately designed regarding sustainability. Important factors that promote sustainable software include design principles, coding principles, user experience, non-technical aspects, like effective documentation, training, and funding. Besides software requirement prioritization and security, reproducibility is one of the design aspects. It is closely related to version controlling. For this purpose web-based version control repository hosting services, like GitHub, Gitlab, and Bitbucket may be used.\footnote{https://bitbucket.org/product/, https://github.com, https://about.gitlab.com}

The requirement of sustainability has to be met by both closed and open source software. Despite the considerable importance of open-source software in science, the requisite attention is often not payed to their sustainability and development. Funding requests are not granted due to a lack of sufficient impact, though writing user-friendly software for a wide range of data requires more effort and thus funding. For instance, US National Science Foundation (NSF) refused a funding proposal for the Python ecosystem, only five days after the publication of the black hole image, that was generated by using Matplotlib and other open-source Python components [Nowogrodzki 2019]. Here, the recognition of research software as key component of the research process could open up a path to a greater (financial) support.

An important prerequisite to raise the impact of research software, is to increase its visibility, so that other researcher may find it. To simplify the processes of finding research elements, Wilkinson et al. [2016] formulated guiding principles and practices that apply on data and their associated algorithms, tools, and workflows.

\subsection*{2.3.3 FAIR Guiding Principles}

The FAIR Guiding Principles are explicitly not prescriptive standards, but may serve as a basis for valuable standards [Mons et al. 2017]. In this request, various stakeholders from the academic, industry, and scholarly publishing sector agree funding agencies, that are actively engaged in the development process. Their claim resulted in the statement “that all research objects should be Findable, Accessible, Interoperable and Reusable (FAIR) both for machines and for people” [Wilkinson et al. 2016, p. 3]. These claims are compiled
2.3. Good Scientific Practice

**Box 2 | The FAIR Guiding Principles**

<table>
<thead>
<tr>
<th>To be Findable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1. (meta)data are assigned a globally unique and persistent identifier</td>
</tr>
<tr>
<td>F2. data are described with rich metadata (defined by R1 below)</td>
</tr>
<tr>
<td>F3. metadata clearly and explicitly include the identifier of the data it describes</td>
</tr>
<tr>
<td>F4. (meta)data are registered or indexed in a searchable resource</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To be Accessible:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. (meta)data are retrievable by their identifier using a standardized communications protocol</td>
</tr>
<tr>
<td>A1.1 the protocol is open, free, and universally implementable</td>
</tr>
<tr>
<td>A1.2 the protocol allows for an authentication and authorization procedure, where necessary</td>
</tr>
<tr>
<td>A2. metadata are accessible, even when the data are no longer available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To be Interoperable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.</td>
</tr>
<tr>
<td>I2. (meta)data use vocabularies that follow FAIR principles</td>
</tr>
<tr>
<td>I3. (meta)data include qualified references to other (meta)data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To be Reusable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1. (meta)data are richly described with a plurality of accurate and relevant attributes</td>
</tr>
<tr>
<td>R1.2. (meta)data are released with a clear and accessible data usage license</td>
</tr>
<tr>
<td>R1.3. (meta)data meet domain-relevant community standards</td>
</tr>
</tbody>
</table>

Figure 2.5. The FAIR Guiding Principles [Wilkinson et al. 2016, p. 4]

in Figure 2.5. Also the German Research Foundation [Deutsche Forschungsgemeinschaft 2019] refers to the publication of research data and principal materials according to these principles in their thirteenth guideline.

The Accessible Principle is not intended to mean Open Access. For reasons of competitiveness, national security, or personal privacy, it may be required that data are not publicly available. To meet the specifications of the FAIR Principles, it is sufficient to provide metadata containing information about their context of generation and the conditions for access and reuse [Mons et al. 2017]. The FAIR Guiding Principles are characterized as a continuum, so that research objects, within the bounds of possibility, should be designed as FAIR Digital Objects as far as possible. Figure 2.6 outlines the Principles in a hierarchical order. Panel A indicates the worst case as far as the data cannot be used once more. Such a flaw is inherent in unstably linked supplementary data that were not characterized by a Persistent Unique Identifier (PID). This applies to more than 80% of the data sets in the current research practice. In ecology and evolution different causes for re-useless data resulted by inspecting the publicly available data sets of 100 studies. Therein, 56% were found incomplete, and in 64% of the studies, the type of archives did not permit reuse. Roche et al. [2015, p. 7] identified “inadequate metadata, the use of proprietary and non-machine-readable file formats (e.g., data tables archived as PDF and word documents; […]), and failure to archive raw data” as main obstacles for the reuse of data.
The first step towards FAIRness is the allocation of a PID (panel B). The metadata of a research object contain intrinsic and user-defined provenance information (panel C). The intrinsic metadata include the information added by the data generation process, for instance, the file format, time stamp, and other features. The FAIR research objects with a restricted element (panel D) are followed by well-defined conditions that are publicly available elements (panel E) on the way to maximal FAIRness. The objective is an Internet of FAIR Data and Services (panel F). In this state, all FAIR data are interconnected and the required minimum for the participation of data is the FAIRness level C.

The FAIR Guiding Principles give the same priority to machine-driven activities, due to their significant impact on handling the large amount of scientific data in the global data ecosystem. Abstracting from domain-focused directives for data management and archiving, FAIR focuses on domain-independent guidelines, applicable for many different research outputs. Intentionally FAIR refrains from technical requirements to facilitate the expansion of reuse by various implementations. Since the focus is clearly on findable, accessible,
and interoperable research objects resulting in their reusability, various approaches for the realization guided by the FAIR principles are welcome. For instance, the Resource Description Format (RDF) meets the requirements of the FAIR Principles and is frequently used, but is not a suitable solution for every research object. So, more applicable concepts may emerge [Mons et al. 2017].

Some Platforms that are listed in Table 2.1, already translate the FAIR Guiding Principles into practices [Wilkinson et al. 2016]. GitHub is not rated as a suitable archival repository, as the data preservation may not be assured regarding by the owner deleted repositories [Barba 2019].

Table 2.1. FAIR following platforms, in accordance to [Wilkinson et al. 2016] and [Barba 2019]

<table>
<thead>
<tr>
<th>FAIR following Platform</th>
<th>Application</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataverse</td>
<td>All research fields</td>
<td>dataverse.org</td>
</tr>
<tr>
<td>Dryad</td>
<td>All research fields</td>
<td>datadryad.org</td>
</tr>
<tr>
<td>FAIRDOM</td>
<td>Systems Biology</td>
<td>fair-dom.org</td>
</tr>
<tr>
<td>Figshare</td>
<td>Scientific data</td>
<td>figshare.com</td>
</tr>
<tr>
<td>ISA</td>
<td>Life Sciences</td>
<td></td>
</tr>
<tr>
<td>Open PHACTS</td>
<td>Drug Discovery</td>
<td>openphacts.org</td>
</tr>
<tr>
<td>wwwPDB</td>
<td>Protein Data Bank</td>
<td>wwpdb.org</td>
</tr>
<tr>
<td>UniProt</td>
<td>Protein sequences</td>
<td>uniprot.org</td>
</tr>
<tr>
<td>Zenodo</td>
<td>All research fields</td>
<td>zenodo.org</td>
</tr>
</tbody>
</table>

2.3.4 Open and FAIR Research Software

The envisaged state of a new scholarly communication corresponds to a dense and searchable network, in that all research objects are uniquely identified and interconnected [Wofford et al. 2020]. This state of maximal FAIRness, that research objects like publications, code, data sets, and documentation can reach, was represented by panel F in Figure 2.6.

Considering the FAIR Guiding Principles, Jiménez et al. [2017] made four easy-to-implement recommendations, that should promote the application of existing best practices resulting in more discoverable, reusable and transparent scientific software. From the authors’ point of view, their recommendations are less granular and ought to help understand and uptake the best practices. With the support of the community registries the provided software metadata can be adopted to the FAIR Guiding Principles by defining guidelines. The target audience are not only software developers, but also research funders, research institutions, journals, group leaders, and managers of projects producing research software.

The first recommendation suggests to offer a code publicly accessible from the very beginning. Moreover, the code should be easily found, as stated in the second recommendation. This can be ensured by announcing the software metadata in a popular community registry, like bio.tools, biojs.io, Omic Tools, Data Cite, or Crossref. To specify the use,
2. Foundations and Technologies

modification, and redistribution of the source code, a suitable Open Source License should be selected. In the third recommendation an Open Source License is suggested that is approved by the Open Source Initiative (OSI). Open Source does not mean to integrate the developer community into the development process. Therefore, the last recommendation is to define the contribution, governance, and communication processes clearly and transparently.

Hasselbring et al. [2019] formulated their recommendations for FAIR and Open research software more precisely, as described in the following.

**Findability** One core element is the ability to cite the applied research software. But up until now there is no universal standard for citing research software defined, due to the lack of citation standards [Howison and Bullard 2016]. For the attribution, credit, and discovery of research software Smith et al. [2016] formulated the six software citation principles: importance, credit and attribution, unique identification, persistence, accessibility, and specificity. These principles are derived from the data citation principles created by a FORCE11 working group. Citable software requires an uploaded software release, for instance, on figshare or Zenodo, where a DOI is assigned to it. The necessary metadata are noted in a specific file in the root directory. This may be either a CITATION.cff file (Citation File Format) [Druskat et al. 2019] or a machine-readable CITATION.jsonld (JSON’s linked data) file [Katz and Smith 2015].

**Accessibility** For reproducibility it is important to access a specific version of research software. As mentioned in Section 2.3.3, version control services are insufficient for this objective, but required in the development process of software. When the software is in an archiving worthy state, the general-purpose open-access repository Zenodo provides the opportunity to archive the released version and a DOI is assigned to each new version of the software.

**Interoperability** Ideally, research software has a modular structure allowing the reuse of its individual components. This requires appropriate interface definitions.

**Reusability** This principle requires portable research software. This may be realized by using Docker containers or online services, like ASTRO Data Lap, Binder, or Jupyter nbviewer. These services are designed to share Jupyter Notebooks, that are a popular tool in various scientific disciplines. Project Jupyter aims to promote open-source software, open-standards, and services for interactive computing. Therefore, an open-source web application, named Jupyter Notebook, is provided, that promotes the combination of narrative text, life code, equations, and visualizations in one document. Various programming

---

14 https://jupyter.org
languages are supported and a web-based interactive development environment, named Jupyter Lab is made available.

To improve the quality of notebooks, Pimentel et al. [2019] created best practices for their development. These include the following eight principles:

1. Use short titles with a restrict charset (A-Z a-z 0-9 _-) for notebook files and markdown headings for more detailed ones in the body.

2. Pay attention to the bottom of the notebook. Check whether it can benefit from descriptive markdown cells or can have code cells executed or removed.

3. Abstract code into functions, classes, and modules and test them.

4. Declare the dependencies in requirement files and pin the versions of all packages.

5. Use a clean environment for testing the dependencies to check if all of them are declared.

6. Put imports at the beginning of notebooks.

7. Use relative paths for accessing data in the repository.

8. Re-run notebooks top to bottom before committing.

2.3.5 Observatories in Research

As an improved access to research software, [Hasselbring et al. 2020a] propose the application of Research Software Observatories. In general, such observatories contain metadata for classification and citeability of the relevant software. So far, this concept is not implemented practically. For its realization there exist three different design approaches for the design.

First proposal: the Research Software Observatory allows the sharing of the software itself and the exchange of information relating to this software and its status.

Second proposal: the Research Software Observatory is a container for the software following the FAIR and open science principles, so that this software can be used by others.

Third proposal: the Research Software Observatory is a catalog, providing all necessary information about the software, like citation link and address.

These concepts are based on the theoretical reference architecture for Data Observatories, introduced by Tiropanis [2019] and reproduced in the present Figure 2.7. These future Data Observatories aim to provide a catalog of datasources and observations on a portal. Datasources represent files, databases or any query interface that can provide data to be evaluated. The notion observation encompasses analytic applications, visualizations, statistical analyses or other processing of data to generate new findings and knowledge.

These two types of resources are described by structured metadata, containing information about provenance, intellectual property, licensing and overall quality of shared resources.
2. Foundations and Technologies

<table>
<thead>
<tr>
<th>Inter-portal communication layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-data exchange and search, negotiation for trust, reporting breaches</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Portal layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction with publishers, data-subjects, users</td>
</tr>
<tr>
<td>Identifiers and meta-information on users, datasources, observations</td>
</tr>
<tr>
<td>Implementation and monitoring of DOP-agreements</td>
</tr>
<tr>
<td>Negotiation of Access-agreements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Control layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication, Reverse proxy, Access control services</td>
</tr>
<tr>
<td>Application Key generation and management services</td>
</tr>
<tr>
<td>Implementation and monitoring of Access-agreements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authentication sublayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Access to remote authentication services (e.g. OpenID Connect)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authorisation sublayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorisation of authenticated users to access and query datasources and observations as per Access-agreement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obfuscation sublayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering obtained datasources and observations per Access-agreement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases, Datasources, Application servers</td>
</tr>
</tbody>
</table>

Figure 2.7. A reference architecture for Data Observatories [Tiropanis 2019, p. 6]

One key policy of the Data Observatories is the decentralization in terms of decoupling data (datasources) from analytics (observations), that is designed in consideration of identifiers, meta-information, access control, consent, portals, decoupling of catalog from storage, inter-portal communication, and Application Programming Interfaces (APIs).

2.4 Technologies

**MongoDB**  MongoDB is a document-based, general purpose, NoSQL database built. It operates on JSON-like documents.

**Scrapy**  Scrapy is a free and open-source webcrawling framework, written in Python. The core elements are self-contained crawlers, also named spiders. They define how a website is crawled and the data are extracted. After a spider has requested a specified start URL, the response is parsed with selectors that are build over the libxml2 library.

---

15 https://www.mongodb.com
16 https://scrapy.org
Chapter 3

Analysis of Research Software

Due to ongoing discussions on reusable research software and its publication, as well as on the reproducibility of its findings, Hasselbring et al. [2019, 2020b] conducted a pilot study. Its subject under investigation was research software, whose current publishing state and sustainability was discovered. In an initial step, research software has to be identified. This procedure is based on references, either to research software in a research publication or to a research publication in research software. Consequently, research publications and repositories for research software form the data basis of the study. This basis is restricted to software repositories hosted on GitHub and to research publications published in the ACM digital library (ACM) or on the arXiv e-print repository (arXiv).  

If a GitHub software repository contains a Digital Object Identifier (DOI) of a research publication, the repository is ranked as research software, as well as GitHub repositories mentioned in a research publication. For the analysis, it is allowed that the links between research publication and software repository are not bidirectional. The resulting sample size considered in the pilot study amounted to over 5,000 software repositories. The main part of these software repositories, in total 2,872, are cited in publications on arXiv, whereas 1,204 publications in ACM mention a repository, and 1,092 GitHub repositories contain a DOI. The sample is quantitatively analyzed with regard to the research areas and software sustainability. To identify different categories of research software and relationships between publications and software, a subsample of the ACM publications is qualitatively evaluated.

3.1 Research Questions

This study intends to implement the pilot study again and verify its results. For this purpose, the links between research publications and software repositories are examined in detail. Like the pilot study, the research scope encompasses the investigation of research publications, released in ACM or on arXiv, as well as software repositories hosted on GitHub. A software repository is considered as research software if it contains a valid DOI reference of a research publication. The required references between research software and research publications have two different forms, depending on the direction. Considering the source code repositories, a link to a research publication is indicated by a DOI. And in research publications, a reference to a software repository is expressed as a github.com/
3. Analysis of Research Software

{owner}/{repository} URL. The harvested metadata of the identified research software are quantitatively and qualitatively analyzed to answer the following three research questions, derived from the pilot study.

RQ1: How frequent are links to and in research software in specific research areas?

This research question is motivated by the understanding whether in specific research areas referencing research software is more frequently applied. The research area is determined by means of the associated publications. Their DOI allows the request of metadata, that contain an International Standard Serial Number (ISSN). By means of this information the subject of the publication medium, the publication is released in, can be derived. The journal subject is assigned to the research software repository and thus specifies its research area. These data are evaluated descriptively.

RQ2: What is the current state of research software sustainability?

Sustainable software is defined by Hasselbring et al. 2019 as live software with a greater lifespan. The criterion live is satisfied, if the last commit was made within the last 12 months, otherwise the research software repository is dormant. The last commit is also used to compute the lifespan of research software, which is specified as the period between the first and the last commit. Like for the above described research question, a descriptive analysis is applied to these data.

RQ3: How is research software linked to its referenced research publication?

Research software is developed to address various tasks in a research project. Hasselbring et al. [2019] identified, for instance, software as a research result, both privately developed and collaboratively constructed and maintained, or as an object of study itself. Therefore, a closer look will be taken on the relationship between research software and its related research publication. Another aspect concerns the analysis of the research software characteristics. The referenced software may be used as framework within the research project or it may be mentioned as related work [Hasselbring et al. 2019]. Since these analyses are of qualitative nature, the sample is restricted to a limited number of GitHub software repositories, that are referenced in ACM publications.

3.2 Requirements and Constraints

The application, developed to identify and analyze FAIR and Open research software, as described by Hasselbring et al. [2020a], must meet miscellaneous functional and non-
3.2. Requirements and Constraints

The functional requirements of the application describe its functionalities provided for a user and its behavior. Each requirement is specified as a use case, represented in a Use Case Diagram, see Figure 3.1.

**UC01: Run Study**  The execution of the complete study combines the data sampling and analysis processes, more precisely the use cases Identify Research Software and Analyze Research Software Metadata. This use case should not necessitate special knowledge, but should instead be executable by all interested third parties with and without deeper understanding of software technology.

**UC02: Identify Research Software**  Potential research software candidates are identified by inspecting the metadata of repositories and publications. The relevant data may be received in two ways. Either they are obtained via the Access Data use case. Or they are gathered by means of the Harvesting Repositories and Harvesting Publications use cases. The corresponding harvester use case is addressed if the condition of the extensions points repositories and publications is satisfied. In other words, new repositories, new publications, or both are required. These two ways may be combined to complement existing data with new ones. The recognized research software is stored in an appropriate format.

**UC03: Analyze Research Software Metadata**  This use case receives its data from the Access Data use case. The obtained details about research software are analyzed regarding their field of research and the sustainability of their repositories. For the analysis the research software is classified into the three groups GitHub, ACM, and arXiv by the source of the repository reference. The set, named GitHub research software, involves all repositories containing a valid DOI. The ACM research software set encompasses the repositories that are referenced in publications released in ACM. The repositories, referenced in a paper published on arXiv, are grouped together as the set called arXiv research software. For each set, the distribution of the corresponding research areas will be
3. Analysis of Research Software

Figure 3.1. The Use Case Diagram for the application to determine the current state of research software publication practice and the sustainability of research software
3.2. Requirements and Constraints

presented, whereby for the arXiv research software set the research area Computer Science will be considered in more detail. Also the average lifespan of these three sets will be analyzed, as well as the quantity of life and dormant repositories.

**UC04: Mining Repositories**  Software repositories hosted on GitHub are the data basis of this use case. The user wants to retrieve all repositories including one or more specified keywords. In this study, repositories are of interest if they contain a DOI reference to a research publication. The search period should cover the period from 01.01.2008 until the specified reporting date. Regarding the reusability of research software, the user may use this subroutine for other objectives than defined in this study by modifying keywords and the search period.

**UC05: Mining Publications**  The data basis for this use case is provided by the papers published in ACM and on arXiv. The user wants to gather all metadata of publications that contain a link to a repository hosted on GitHub. In the context of reusability, the keywords may be modified.

**UC06: Access Data Set**  The user or other subroutines want to retrieve data, that is created by other use cases or added externally.

**UC07: Archive Data Set**  If the data set is considered worthy of preservation, the user can archive it, for instance on Zenodo. So, it will be made available for validation and further analysis.

**UC08: Reproduce Study**  The reviewer of this study wants to reproduce the findings with newly collected data. This requires the execution of the use case Run Study.

**UC09: Reproduce Analysis**  The reviewer wants to reproduce the results of this study with the provided data set. Therefore, he runs the use case Analyze Research Software Metadata.

### 3.2.2 Non-Functional Requirements

The non-functional requirements refer to the quality and constraints of the application. Concerning the product quality of software, the ISO/IEC 25010 specified a quality model with eight quality characteristics. These characteristics are considered in the design process, as well as two constraints.
3. Analysis of Research Software

Quality Requirements

The quality characteristics of the ISO/IEC 25010 include supportability, performance, portability, reliability, functionality, compatibility, security, and usability. Due to the open science principles, particular attention is paid to the realization of the usability, maintainability, and portability for this application.

Constraints

No Persistent Data  This study provides a snapshot of the current state regarding the research software development and publication behavior. The data obtained may change over time because of added or deleted references and repositories. Hence, the results of reproduction studies may slightly change if they are conducted with newly created data.

Lack of Application Programming Interfaces (API)  Some digital libraries and meta search engines for research publications provide APIs that allow data mining. This prerequisite for harvesting research publications is not satisfied by all of them. The lack of an appropriate API impedes the compliance with the requirements of reproducible research.

3.3 Framework Architecture

The non-functional requirements, described in Section 3.2.2, are decisive factors in the design process of the framework architecture, mainly focusing on great usability, reusability, and maintainability. In order to meet these requirements, the open-source web application Jupyter Notebook [Kluyver et al. 2016] was chosen as framework. For the development process JupyterLab, version 2.2.8, is used as development environment in combination with the programming language Python, version 3.6.9 [Van Rossum and Drake 2009]. More detailed information about the used packages and their versions are specified in a requirements file, included in Appendix B.

In accordance with the use cases, the core elements of the application are six Jupyter Notebooks (hereafter referred to as Notebooks; number 1, 2, 3, 4, 5, 6 in Figure 3.2), running on a Jupyter Notebook Server, and two harvester modules (7, 8), as shown in Figure 3.2. These components are supplemented by a configuration file (A), a MongoDB database (C), version v3.6.3, and two Python modules for the database functionality (D) and auxiliary functions (E). These elements are described in the following.

3.3.1 Configuration File

To facilitate the extension of the individual Notebooks and to run them without the need of changes within the cells, the Notebooks get their parameters from a configuration file (A), that is included in Appendix A. For its implementation the configuration file format YAML
3.3. Framework Architecture

Figure 3.2. The Component Diagram for the framework to analyse the current state of the publication practice and sustainability of research software
3. Analysis of Research Software

is used. The parameters are loaded with the YAML parser PyYAML, version 5.3.1. Various parameters may be specified. For instance, the keywords that should be contained in the repositories and publications. Further parameters define the repository hosting services, e-print repositories, and digital libraries to be harvested. Also individual processing steps in the data cleansing and preparation process may be selected.

In addition, the configuration file contains an interface for the available harvester classes, named `supported_sources` (B). As shown in Listing 3.1, it provides the necessary information, like the harvester class names and whether a token is required for the API calls. When adding further harvester classes for repository and publication hosting services, a new entry is attached to this list. It is located in the configuration file, as all Notebooks can access it. Another option would be the definition of the interface in each individual Notebook. In that case, it should be noted that the variable is overwritten, when another Notebook that also contains this variable, is started from the current Notebook.

Listing 3.1. Interface for the available harvester classes

```python
supported_sources:
    arxiv:
        token_required: False
class: ArXivHarvester
github:
    token_required: False
class: GitHubHarvester
```

3.3.2 Data Model

All data, that are gathered and processed by this application, are stored in a MongoDB database (C in Figure 3.2). The database and its database tables are specified in the configuration file. The Class Diagram, see Figure 3.3, presents all database tables with its recorded parameters. The present essay focuses on research software with its associated repositories and artifacts. The research software repositories are managed in the `rs-repositories` database table. Each database entry contains information about the research software repository’s ID, full name, hosting service, language, sustainability features, and assigned research area.

Further metadata can be received from the `repositories` database table, where all metadata are stored that are received from the API of the harvested repository hosting service. Additional database fields provide information about the request date, the hosting service, and the search terms, responsible for the admission to the data set. As an optional information the Readme file of a repository may be included. Each research software repository is linked by its ID with exactly one repository (composition 1 in Figure 3.3).

---

²https://pypi.org/project/PyYAML/
Conversely, a repository has a maximum of one associated research software repository, since not every repository is a research software repository.

Another database field in the `rs_repositories` database table contains the identifier and its type for the associated research software artifacts. By means of this information, the metadata of a publication can be looked up in the `publications` database table (association 2), provided that the DOI belongs to a harvested publication. The publications metadata contain details about the publications source, publishing and request date, available identifier, and the context of the search term. Not included is information about associated research software repositories. Thus, these details can be obtained with a detour via the `rs_artifacts` database table. A publication is at maximum linked with one artifact by the identifier (association 3), since not every publication contains a link to a GitHub repository and is therefore no research software artifact. And vice versa, a DOI identifies not only

---

Figure 3.3. The Class Diagram for the data model

---

3.3. Framework Architecture

---
3. Analysis of Research Software

publications, but also data sets, software, and other artifacts. Therefore, each artifact is associated via its ID to zero or one publication.

The rs_artifacts database table manages the research software artifacts. An artifact comprises all objects identified by a DOI and publications identified by an arXiv ID. The preferred identifier is the DOI. If the DOI is not available, the arXiv ID is chosen as alternative. Each artifact is associated with an indeterminate number of repositories (association 4), and thus research software repositories (association 5), that can be looked up with the repository’s full name, stored in the artifacts repos list. In return, the linked artifacts for each repository can also be addressed via the repository’s full name. Each research software repository contains the identifiers of its associated artifacts and can address in this way the artifacts.

An optional information are the DOI metadata, as well as the assigned research area. In the case, that the DOI metadata are available and contain an ISSN or ISBN, in the associated publication_subjects database table the subject of the container, the artifact is part of, may be looked up (aggregation 6). Container is an umbrella term for journals, books, and conference proceedings. Artifacts, identified by an arXiv ID, usually have a primary category that allows the lookup of the arXiv subject in the associated arxiv_subjects database table (aggregation 7).

The subject of an artifact is an integral part of this study and is derived from the primary arXiv category, the container ISSN, ISBN, or title. In the arxiv_subjects database table, the arXiv category taxonomy is complemented with a super discipline according to the All Science Journal Classification (ASJC) system of Scopus. This classification system is also the basis of the publication_subjects database table. It is composed of the journals, books, and conference proceedings listed in the Scopus source list and the book title list.³

3.3.3 Database Module

Figure 3.4 illustrates the Class Diagram for the database module (D in Figure 3.2). The _Database is only visible within this module. It initializes the database connection for the Collection class to establish a connection to a specified database table. The Collection methods for the data management are based on the basic methods provided by the _Database class. Thus, the underlying database become easily interchangeable by modifying the _Database class.

Some operations on the data of the repositories, rs_repositories, and rs_artifacts database tables have a more complex structure and consist of a number of basic methods. Since these operations are adjusted to the individual database table, separate classes provide the required methods. Due to the inheritance from the Collection class, also the basic methods can be used.

³https://www.scopus.com/home.url
3.3. Framework Architecture

![Figure 3.4. The Class Diagram for the database module](image)

### 3.3.4 Auxiliary Functions

Besides the functionality of the harvester methods, functions are required that may not be assigned to specific classes or that are called by various Notebooks. These functions are provided by a separate Python module for auxiliary functions (E in Figure 3.2). For instance,
3. Analysis of Research Software

![PublicationHarvester interface](image)

**Figure 3.5.** The PublicationHarvester interface and the ArXivHarvester class

the function for the DOI extraction is located here. A given text fragment is searched by using the Python re module for regular expression matching operations [Van Rossum 2020]. For this purpose, the two regular expressions are formulated, covering valid DOIs and shortDOIs:

\[
\begin{align*}
\text{r'}(10 ?\cdot ?[0-9])((?:\cdot[0-9]+)+ ?/ ?[-_.;():A-Za-z0-9]+)'
\text{r'}((?:doi\.org| 10|doi\.org/10|doi ?: ?10)/(?!10)[a-zA-Z0-9]+)['\s><.)']
\end{align*}
\]

### 3.3.5 Harvester Modules

The harvester modules (F, G in Figure 3.2) provide all obligatory methods for the repository and publication harvest processes. The required methods are defined in the RepositoryHarvester interface, see Figure 3.6, respectively in the Publication Harvester interface, see Figure 3.5. For the research software identification process are also harvesting methods used to gather further information, like the first commit date. These mandatory methods are specified in the Repository Complementer interface, see Figure 3.6. Due to the number of required methods and potential additional harvester classes, implementing these classes in the corresponding Notebooks leads to complex and cluttered Notebooks. So for greater clarity, the harvester classes are implemented in independent Python modules.

With the instantiation of the ArXivHarvester class (F) the values of the start index, the results per iteration, and the sleep time after each request can be passed. The default values are zero for the start index, 500 results per iteration, and 5 seconds to sleep. When instantiating the GitHubHarvester class (G), an authentication token can be passed (character A in Figure 3.16). After the successful validation of the token, it is set to the HTTP(S) header Authorization field for the requests. As recommended by GitHub, the Accept header field is set for every request to ‘Accept’: ‘application/vnd.github.v3+json’ for requesting explicitly version v3 of the REST API, also without a token.\(^4\) Provided that another specific

\(^4\)https://docs.github.com/en/free-pro-team@latest/rest/overview/resources-in-the-rest-api

36
3.3. Framework Architecture

Figure 3.6. The GitHubHarvester class and the RepositoryHarvester and RepositoryComplementer interfaces

header is added to the request, the GitHub REST API appends the text fragments metadata to the response, named text_matches. It contains the context and location of the matching search term in the name and description, but this applies not for results in the Readme file. Testing this feature has shown, that in some cases the text match is too short and the end of the DOI is cut of. Therefore, this option is not used. Instead, the description, received as part of the gathered metadata, is used for the DOI extraction. Depending on the existence of an authentication token, the constructor sets the sleeping time. In the case of a provided authentication token, the search rate is limited to 30 requests per minute and the core rate to 5,000 requests per hour. Without a token, ten search queries can be requested per minute and 60 repository queries per hour. The sleep times are defined by dividing the time unit
3. Analysis of Research Software

![Diagram of preparation steps](image.png)

Figure 3.7. The Activity Diagram for the general preparation steps of all Jupyter Notebooks

by the rate limit, resulting in a search sleep time of two seconds (six seconds without a token) and a core sleep time of 0.7 seconds (60 seconds without a token).

3.3.6 Jupyter Notebooks

The core elements of the framework are six Notebooks, see Figure 3.2. Five of them may be executed independently, supposing that the required data are provided. Only the Study Reproducer Notebook (6 in Figure 3.2) relies on the Research Software Identifier (4) and Research Software Analyzer (5) Notebooks, as the whole study is executed. To run the other two Notebooks the Study Reproducer (6) uses the the built-in magic command `%%run` and is thus the implementation of the use cases **UC08: Reproduce Study** and **UC01: Run Study**.

Between the separate Notebooks no data is exchanged. The Research Software Identifier Notebook (4) obtains its required data from the MongoDB database, where also the results of the processing procedure are stored. If new data should be harvested or be added to an existing data set, this Notebook runs the Repository Harvester (3) and the Publication Harvester (1) Notebooks to gather the required data and store them in the corresponding MongoDB database tables. Like the Research Software Identifier (4), the Research Software Analyzer (5) obtains its needed data from the MongoDB database. By means of the Python libraries numpy [Oliphant 2006], pandas [McKinney 2010], and matplotlib [Hunter 2007] the research software data are analyzed. This Notebook realizes the use case **UC03: Analyze Research Software Metadata**.

The repository hosting services and digital libraries that should be harvested, are specified in the configuration file (Section 3.3.1). To start the harvesting process, the Repository Harvester (3) and the Publication Harvester (1) Notebooks instantiate the corresponding harvester classes.

Before the processing starts, each Notebook, except the Study Reproducer Notebook, loads its required parameters from the configuration file (A) via the PyYAML package, as shown in Figure 3.7. If this file is not located in the same directory or is named differently than `config.yaml`, it cannot be found. An error message is caused and the execution of the current and following Notebook cells terminates. The parameters vary in dependence of the corresponding Notebook. Repository Harvester Notebook (3) expects details about the time interval size to partition the GitHub REST API requests, and the start and end date...
of the search period for the repository acquisition. Also the search terms, the repository hosting services, and the information whether the Readme files of the repositories should be harvested are required. Listing 3.2 illustrates the configuration file fragment for the search period, the time delta, and the tasks to be executed. The Publication Harvester Notebook (1) only needs information about the search terms and the publication source to be harvested. For the processing procedure of the Research Software Identifier (4) the individual steps can be activated or deactivated.

Listing 3.2. Configuration file fragment specifying the search parameters

```
# set the dates for the repository search period
start_date: "2020-10-01"
end_date: "2020-10-01"

# set the repository search interval in the format years|months|days=int
delta: days=7

# specify the information to be collected by the repositoryHarvester
repoHarvester:
  - search_repos
  - readme
```

The Publication Harvester Notebook (1), the Repository Harvester Notebook (3), and the Research Software Identifier Notebook (4) have a more complex architecture than the other Notebooks. Therefore, these three Notebooks are described in greater detail.

### 3.4 Publication Harvester

An important factor for the identification of research software are the research publications. Therefore, this Notebook (1 in Figure 3.2) represents the realization of the use case **UC05: Harvest Publications**. Like in the pilot study [Hasselbring et al. 2019; 2020, b], the research publications are gathered from ACM and arXiv. Whereas arXiv provides an API, the data acquisition of the ACM requires a special handling. Due to the missing API, the ACM website has to be crawled. As this is not a suitable method for reusable software, the ACM is excluded from the Publication Harvester and is implemented in an individual Notebook, named **ACM Crawler (2)**.
3. Analysis of Research Software

![Publication Harvester Activity Diagram](image)

**Figure 3.8. The Activity Diagram for the “Publication Harvester” Notebook**

### 3.4.1 Search Term

The publications, containing a link to a repository hosted on GitHub, are of interest for the further process of the study. These references are formulated in several ways. To cover URLs with the prefixes “http://” and “https://” or without them, `github.com` is chosen as keyword.\(^5\) And it narrows the results by excluding references to GitHub itself, which would be the case for using only `github` as search term. Enclosing the search term in double quotation marks showed no difference in the quantity of results for both ACM and arXiv.

### 3.4.2 Design

In regard to their different realizations, the ACM Crawler and the Publication Harvester are designed separately from each other.

**Publication Harvester**

Subsequent to the preparation steps, see Section 3.3.6, the associated harvester class for the first source is looked up and instantiated, as depicted in Figure 3.8. Each of these classes provides its harvesting method for publications. This includes the construction of the API calls, requesting and parsing the data, and the insertion of the extracted metadata into the database table. Consecutively each source is harvested regarding to all the indicated search terms.

**ACM Crawler**

In contrast to the other Jupyter Notebooks, the ACM Crawler merely requires the name of the database and the associated database table name to insert the publications. After the completion of the preparation steps, the ACM website is crawled, see Figure 3.9. For this purpose one or more start URLs are specified. During the parsing process the required information for each publication are extracted by means of XPath selectors and inserted to the Publications database table.

---

\(^5\)Parameter values and search terms are presented in italics.
3.4. Publication Harvester

3.4.3 Implementation

Like in the preceding Section 3.4.2, the implementation of the Publication Harvester and ACM Crawler are described separately for each Notebook.

Publication Harvester

As defined in the preamble of the research questions, see Section 3.1, in this study the source of the research publications is restricted to arXiv. As further digital libraries, their operating principles, and response formats are unknown at this stage, the complete harvest process is located in the ArXivHarvester class. So this Notebook acts as coordinator and caller for the publication harvester classes, as illustrated in Figure 3.10. By iterating over all sources, the corresponding harvesting method is called for all specified search terms.

In the ArXivHarvester class, see Figure 3.5, the implementation of the harvester method is based on the provided arXiv examples for parsing and paging.\(^6\) Initially, the quantity of publications is requested to address the paging mechanism, as shown in Figure 3.11. The number of total results per single call is restricted to 30,000 publications and one page contains up to 2,000 publications.\(^7\) To step through the publications set, a start value indicates the index of the first result to be returned. The value for the results per iteration indicates the number of publications per page. To avoid a considerable load on the server and to comply with the note to request smaller slices if the number of results exceeds 1,000 results, the number of results per iteration is set to 500.\(^8\) The start indices for the pagination are picked from the range of the first starting index to the number of total results. To

---


\(^7\)https://arxiv.org/help/api/user-manual#paging

\(^8\)https://arxiv.org/help/api/user-manual#calling_the_api
3. Analysis of Research Software

Figure 3.10. The Sequence Diagram for the Publication Harvester Notebook

retrieve the next index, the current index is incremented by the number of results per iteration.

After setting the pagination parameter, the current query is composed and requested. Like in the examples, this function uses the feedparser library, version 6.0.1, to handle the arXiv API response in Atom 1.0 format. As important as the metadata of each entry, are the referenced GitHub repository names and owners. These are located in the summary, the summary details, and the arXiv comments of a publication. Provided that these attributes are present, they are inserted together with the metadata in the publications database table.

ACM Crawler

This Jupyter Notebook is based on the open-source web-crawling framework Scrapy [Kouzis-Loukas 2016], version 2.2.0, to gather the required publications of ACM. The data basis is the ACM Guide to Computing Literature. This bibliographic database comprises, besides the ACM’s full-text articles, books, theses, technical reports, and other documents. The ACM Crawler is not intended to be part of the reusable framework package, and therefore the keyword is set as constant in the starting URLs. Also a search period is not required, as the entire publication period of ACM is searched. Thus, the ACM Crawler only needs to load the name of the corresponding database table and connect to it. In Figure 3.12 the sequence of events in the crawling process is illustrated. After instantiating the CrawlerProcess and

---

9https://pythonhosted.org/feedparser/index.html
passing the current spider, the crawling process is started. The starting URLs and the parsing function are defined in the spider class SpiderACM.

On September 15th, 2020, the total number of publications containing the search term github.com is requested. To manage the quantity of approximately 23,000 publications, different search intervals are defined for the search period from 2008 to 2020. January 1st, 2008 is chosen as starting date, as there are no publications, containing the defined search term, published before this date. ACM offers the search for publications with release dates going back to 1936. When specifying the start URLs, search intervals are chosen that do not exceed 2000 publications, as ACM limits the search results to the most relevant 2000 publications. The intervals 2008 to 2012, 2013, 2014, and 2015 are covered by one starting URL. The first half of 2016 defines one interval and from July 1st, 2016 the start URLs are specified on a quarterly basis. The starting URL is passed to the Scrapy engine and a response is returned to the SpiderACM.

In the parsing process, the XPath selectors are extracted from the response, whereby each selector corresponds to one ACM publication. By iterating over the papers, the
metadata and the search term context are derived. On the website, the context is located in a drop-down box, named Highlights. This box is divided into the two parts abstract and full text, containing the context of the search term occurrence. The appearance in the references section is not covered. The text fragments from the Highlights box and the gathered data are inserted to the database table. Before crawling the next starting URL, the following page for the pagination is extracted from the response. To avoid the blocking of the IP address, prior to the next website download, the process sleeps for three minutes. This period is freely chosen and, so far, has not provoked the blocking of the IP address. It must be noted, that the starting URLs are not processed immediately one after another, but one starting URL per day. It must be taken into account that for a repeated execution of the crawling process, the kernel has to be restarted. Otherwise a ReactorNotRestartable error is thrown.\footnote{https://github.com/scrapy/scrapy/issues/2594}
3.5 Repository Harvester

The Repository Harvester Notebook (3 in Figure 3.2) implements the use case UC04: Mining Repositories. The prerequisites for this part of the study are the specification of github as software repository hosting service. The search period covers the time between January 1st, 2008 and October 12, 2020, so these dates are the default values for the start (2008-01-01) and end date (2020-01-12). The time interval is set to days=7, to manage the quantity of results per API call. The GitHub REST API returns at most 1,000 results per query, segmented in ten pages listing 100 results. Also the rate limits for different kinds of API requests and special characters, that are ignored in an API call, have to be taken into account when designing the implementation of this Notebook.

3.5.1 Search Term

Gathering potential candidates of research software is the objective of this Notebook, whereby a repository is nominated when it references a valid DOI. To achieve the highest possible yield, it is important to specify adequate search terms. After reviewing 100 random repositories, that include the keyword doi, the following variants of mentioning a DOI are identified:

- As DOI link, with DOI resolver, prefix, and suffix,\(^{11}\)
  
  e.g. http://dx.doi.org/10.1088/…

- As DOI, with prefix and suffix,
  
  e.g. DOI: 10.2118/…

- As part of a BibTeX entry,
  
  e.g. doi = {10.1007/…}

So, a DOI comes across in various forms. Due to the ignored special characters in a GitHub REST API call, these appearances cannot be directly defined as search terms.\(^{12}\) As shown in Table 3.1, requesting the search terms doi.org and doi org results in the same quantity of repositories, as well as DOI: 10, doi:10, and doi={10, confirming the ignorance of the special characters and the case insensitivity. This overview of total results for various search terms also aims to get a first idea of the dimensions involved here. The most results are returned for the keyword doi. The adverse effect of using such an imprecise search term, is the presumably large number of repositories in the search results, that contain the word doi, but not a DOI reference. These include, for example, the number two written in Portuguese (dois), or using it in the repository name, like jmarucha/doi-helper, DoidoYo/DoidoYo.\(^{13}\)

In the subsample, the quantity of these repositories adds up to 43 repositories of the 100 chosen ones, indicating a large overhead for the overall sample. Conversely, the search

\(^{11}\)https://www.crossref.org/education/metadata/persistent-identifiers/the-structure-of-a-doi/

\(^{12}\)Special Characters: . , : / \ ' " = ! ? # $ & + ^ | ~ < > ( ) { } [ ]

3. Analysis of Research Software

Table 3.1. Number of results for various search terms in GitHub REST API calls, requested on 10.08.2020

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Quantity</th>
<th>+in:readme Quantity</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>doi</td>
<td>12756</td>
<td>86811</td>
<td>99567</td>
</tr>
<tr>
<td>DOI: 10</td>
<td>2146</td>
<td>60782</td>
<td>62928</td>
</tr>
<tr>
<td>doi.org</td>
<td>1241</td>
<td>41015</td>
<td>42256</td>
</tr>
<tr>
<td>&quot;doi 10&quot;, &quot;doi:10&quot;, &quot;doi={10}&quot;</td>
<td>917</td>
<td>31278</td>
<td>32195</td>
</tr>
<tr>
<td>&quot;doi.org&quot;, &quot;doi org&quot;</td>
<td>1108</td>
<td>26921</td>
<td>28029</td>
</tr>
</tbody>
</table>

The term “doi org” covers only the DOI links and repositories containing “doi” and “org” next to each other. The same holds for “doi 10”, as it will return only repositories containing “doi” and “10’ side by side. This would lead to a large amount of unconsidered research software candidates, since DOI names and DOI links are approximately equally used as reference, as shown in Figure 3.13b. To make a compromise between these too broadly formulated and a too narrow defined search terms, doi+10 is chosen. It will cover the DOI links, the DOI names, and the BibTeX entries. Since doi and 10 are not necessarily located next to each other, like in the DOI link and the BibTeX entry, the double quotation marks are not used in the search term.

As shown by the table, most of the DOI references are located in the Readme file of a repository. Also in the sub sample, 53 of 56 DOIs are found there, see Figure 3.13a. To retrieve these repositories, a second keyword with the suffix +in:readme is specified in the configuration file, see Appendix A. Github provides various search qualifiers to formulate queries. Among them is the in qualifier to determine the part of the repository to be searched, like the repository name, description, or the readme file. When omitting the in qualifier, only the repository name and the description are included into the search space.14

3.5.2 Design

The Activity Diagram, Figure 3.14, illustrates the sequence of events in the process of harvesting repositories. Subsequent to the preparation steps, see Section 3.3.6, the harvester class for the specified repository hosting service is instantiated, provided that the searching process should be executed. By iterating over the specified search terms one of the required parameters of the harvester method is set. Another parameter is the search interval, computed in accordance to the specified search period and interval size. The results and their metadata are inserted into the repositories database table, complemented with the information about the hosting service, the current keyword, and the request date. The HTTP(S) response header contains two pieces of information, that are necessary for the following requests. To comply with the rate limits, the number of remaining requests is

14https://docs.github.com/en/github/searching-for-information-on-github/searching-for-repositories
3.5. Repository Harvester

![Bar charts](image)

(a) Location of the DOI.  
(b) Appearance of the DOI

Figure 3.13. Location and appearance of referenced DOIs in the 100 repositories of the sub sample

obtained and set as parameter for the next call. As standard, 100 results per page are requested and the pagination information for the next page is extracted from the Link header field. If no further pages are available, the next search interval respectively the next keyword is processed. If these parameters are processed, the data acquisition is continued with the next source.

For further processing, the context, respectively the referenced DOI of a publication, is of interest. Therefore, the Readme file for each repository is requested, when the search for all keywords within the specified sources is completed. This step can be executed independently from the data acquisition process. Therefore, the hosting service (source field in the database entry) is looked up for each repository and its harvester class instantiated. After requesting the Readme file, the database entry is updated. Like in the repository search process, the remaining core rate limit is picked from the header. After each call the process sleeps for the determined seconds.

As recommended by Smith et al. [2016], citable software should provide its metadata in a CITATION or CITATION.jsonld file, see Section 2.3.4. This file may also be used as an additional distinctive feature of research software. But GitHub provides only the in:readme qualifier to find repositories based on its content. The search for other specific content is only supported for single repositories by the file finder or code-specific search qualifiers.\(^\text{15}\) Therefore, this identification variant is not pursued further in this approach.

An alternative source for harvesting GitHub repositories is GH Archive.\(^\text{16}\) The archival period started on February 12, 2011.Hourly an archive of all occurred events, including commits, forks, and commenting, is created. The raw data can be accessed by downloading Gzip files. The recording of all events leads to large archives, therefore this variant is not

\(^{15}\)https://docs.github.com/en/free-pro-team@latest/github/searching-for-information-on-github/searching-for-repositories#search-based-on-the-contents-of-a-repository

\(^{16}\)https://www.gharchive.org
3. Analysis of Research Software

Figure 3.14. The Activity Diagram for the "Repository Harvester" Notebook
used in this study. Moreover, repositories created in the period between 2008 to 2011 should also be harvested. GH Archive provides its hourly updated data also on Google BigQuery. To work with this data, a Google project has to be created that requires a Google account. Moreover, Google BigQuery is a pay service. These conditions of necessary accounts and payments are incompatible with the idea of using this framework without any needed account or authentication. Consequently, this variant is also not used.

3.5.3 Implementation

Like the Publication Harvester, the Repository Harvester coordinates the repository search process. Along with the sheer acquisition of the metadata, it also completes the metadata with the repository Readme file. In the configuration file is specified whether one or both of the two tasks should be executed. Within the preparation steps the configuration file is loaded, as described in Section 3.3.6.

Harvesting Repository Metadata

The sequence of events during the repository metadata harvesting process is illustrated in Figure 3.15. The search procedure iterates over the specified repository hosting services (character A in Figure 3.15), in this case Github. Initially, the GitHubHarvester class is instantiated. Consecutively the keywords (B) and search intervals (C) are passed to the search method (D). The current search interval (C) is obtained from the GitHub Harvester class in the required format for GitHub Search API calls. The start and end parameters indicate the period of time the repositories are created in. The delta parameter describes the time interval length for each request, for instance \texttt{months=1, days=7}. As a large amount of periods may be created, this function is implemented as a generator function. It returns a lazy iterator and does not store the periods in memory. The next interval is implicitly called within a for loop.

The obtained metadata of each repository are inserted to the repositories database table via the database module (E). If the HTTP(S) response header contains links of the enclosing result pages, the URL of the next page is extracted (F) and assigned to the variable for the next URL. If no further result page is available, the returned URL is None. This variable is also set to None, when the HTTP(S) response header does not contain pagination links, as it is the control parameter for the pagination loop (H). Furthermore, this variable is passed together with the current keyword and interval to the search method (D). Provided that the next URL is not None, the results for this URL are returned. Another important information in the HTTP(S) response header is the remaining rate limit. It is extracted and passed as additional information for the search method (D). But before executing the next call, the process sleeps for a specified time (G), that is prescribed by the harvester class.

How the repository search method (D) works, is represented in Figure 3.16, whereby the surrounding loops (A, B, and C) are omitted. To determine the current URL, the passed parameter for the next URL is checked (I). If it is nonexistent, the URL is composed
3. Analysis of Research Software

Figure 3.15. The Sequence Diagram for the Repository Harvester Notebook
3.5. Repository Harvester

Figure 3.16. The Sequence Diagram for the repository metadata harvesting method of the GitHubHarvester class

according to the passed keyword and search interval, and the constant elements, the base URL, the infixes and the suffix. Before sending the constructed API call, the remaining rate limit is checked (2). The number of remaining requests, received from the Repository Harvester Notebook, is forwarded. If this information is unknown (remaining_requests = -1), it is requested from the GitHub REST API (GET /rate_limit). The REST API rate limit will not be decreased by accessing this endpoint.\(^{17}\) In the case of zero remaining requests, the process sleeps until the defined reset time.

After sending the HTTPS request (3), the HTTP(S) response status is checked for redirects and error codes (4). If a response with an error status is received, None is returned. In the case of a redirect, the URL in the Location header field is requested and the response is returned to the Repository Harvester, in the same manner the return value is set for responses with the status code 200 (5).

\(^{17}\)https://docs.github.com/en/rest/reference/rate-limit
3. Analysis of Research Software

Figure 3.17. The Sequence Diagram for the readme harvesting process of the Repository Harvester Notebook
Harvesting Repository Readme Files

The configuration file contains a parameter that indicates whether the corresponding Readme file for each repository should be requested (character A in Figure 3.17). This process iterates over all repositories in the repositories database table without a readme field (B). This condition allows the continuation after intended or unintended breaks. By looking up the hosting service of the repository (source field in the repository entry), the associated harvester class is instantiated (C). After requesting the Readme file, the repository database entry is updated with the HTTPS response body containing the raw file content (D). There are also repositories without a Readme file. Here, the string empty readme is added to the database entry, in order to ensure the stop condition of the loop. To spread the GitHub REST API server load and avoid a long wait if the rate limit is exploited in the first minutes of an hour, the process sleeps for a specified time after each request. This also prevents the time out error of the MongoDB cursor, provoked by inactivity for longer than 30 minutes. Due to an idle session, it may happen, that the load balancer does not maintain the connection between the server and the Notebook (AutoReconnect: localhost:27017: [Errno 104] Connection reset by peer; ServerSelectionTimeoutError).

The Readme files are gathered with the content request method (C in Figure 3.17) of the GitHubHarvester class, illustrated in Figure 3.18. This method is also used for data cleansing and preparation steps, therefore, the expected response category has to be passed as parameter. Supported categories are readme, commits, content, metadata and user. Additional required information are the repository ID and the number of remaining requests. For pagination purposes a URL may be passed, that is treated with priority. After checking the remaining rate limit (1), the current header is adapted to the request of Readme files (2). The content should be sent in raw format instead of the usual json format. Therefore the information application/vnd.github.v3.raw is set in the Accept header field. With the constructed URL (3), the content is requested (4). In the case of a successful request, the response together with the remaining rate limit is returned (6). If the response contains a redirect, this link is followed and the received response returned (5). The return value is set to None, when the status code does not comply with 200, 301, 302, or 307 (7).

3.6 Research Software Identifier

The above described Notebooks, the Repository Harvester (3 in Figure 3.2) and the Publication Harvester (1), are designed for data collection purposes. This is only one part of the research software gathering process, as only the repositories and publications are chosen that meet the condition of containing the specified keywords. Among these repositories are others than source code repositories, for instance, repositories only containing a Readme file or empty ones. Thus, in these steps the research software candidates are determined. The Research Software Identifier Notebook (4) provides the next two
3. Analysis of Research Software

![Sequence Diagram](image)

**Figure 3.18.** The Sequence Diagram for the content request method of the "GitHubHarvester" class
steps in the data acquisition process. On the one hand, the data cleansing to select the research software out of the set of candidates. On the other hand, the data preparation for the analysis. So the main objective of the Research Software Identifier is to create a ready to evaluate research software data set, corresponding to the use case UC02: Identify Research Software. Although this Notebook is the largest one, it consists mainly of operations on the database tables and requests for additional information, like the first commit date, and repository content. For this purposes the GitHubHarvester class provides the corresponding methods that are based on the general harvester method.

3.6.1 Design

Like in the other Notebooks, the Research Software Identifier (A) gets its required parameters from the configuration file, see Figure 3.19. The user can decide whether new publications (A) and repositories (A) should be gathered or existing database tables are processed. The selection process for research software starts by using existing metadata to decrease the quantity of data and thus reducing the number of following API calls for the data cleansing and preparation. One of the criteria to identify a repository as research software is an included DOI name of a research publication. Thus, each repository is scanned for valid DOI names (B). If no DOI is found, the repository is discarded. In contrast, repositories containing a DOI name are inserted into the database table rs_repositories.

The repositories referenced in a research publication complement the rs_repositories database table (C). If a repository is already inserted by the preceding step, the research publication is added to the repositories reference list. In the other case, a new database entry
3. Analysis of Research Software

with the repository’s full name is added to the database table. Besides the references to a specific repository of the form https://github.com/{owner}/{repository}, also owner of repositories are mentioned by a link like https://github.com/{owner}. As subsequent step these links are also extracted from the gathered publications (D). It is separated from the repository’s full name extraction, as GitHub Rest API requests are required to get all repositories of an owner. For each repository it is checked whether an entry already exists in the rs_repositories database table and whether the referencing publication is registered in the repository’s reference list. If this is not the case, the publication is added to the reference list respectively the repository is inserted to the database table. When requesting all repositories of an owner, the repository metadata are also contained in the HTTP(S) response. Thus, the metadata for almost all repositories are harvested, except for the repository names that are extracted from the publications. Therefore, for all entries in the rs_repositories database table the metadata are looked up in the repositories database table (E). In the case of missing metadata, they are requested from the GitHub REST API. Furthermore, research software should contain a large computational component [Kanewala and Bieman 2014]. Repositories that consist solely of a Readme file or do not contain any source code file are not considered as research software. Explicitly checking this property for all existing programming languages may not be implement straightforward. Therefore, a quick check is implemented that discards all repositories only consisting of a Readme file, License file and .gitignore file (F). Just like the repositories, the existence of the DOIs is audited and their metadata are gathered (H). This review process is the last step of the repository cleansing. The repository preparation comprises the harvesting of the first commit dates for the computation of the lifespan (G). The live status of a repository is determined via the last commit date. These data are stored in the just created rs_repositories database table. The last step, is the assignment of the research subject to the research software repositories (I).

3.6.2 Implementation

The Research Software Identifier Notebook deals with large database tables and has to comply with the GitHub Rest API rate limit, resulting in long runtimes. Therefore, each individual step of these identification and preparation processes can be executed solely, selectable in the configuration file, as illustrated in Listing 3.3. As an initial step the user can decide whether new repositories and publications should be harvested (A). If the parameters new_repositories and new_publications are not commented out, the corresponding harvester Jupyter Notebooks are called via the built-in magic command %run. The required data for the data cleansing are retrieved from the Repositories and Publications database tables. Due to the above mentioned runtime, this Jupyter Notebook consists of several small processes that are solitary considered in the following.
3.6. Research Software Identifier

Listing 3.3. Selectable repository and publication harvesting cleansing and preparation steps

```python
# specify the steps to be executed by the rs_identifier
rs_identifier:
# - new_publications
# - new_repositories
  - dois
  - repo_name
  - user_name
  - metadata
  - content
  - commits
  - crossref
  - subject
  - subject_arxiv
```

DOI References

The large data set of repositories is reduced by discarding repositories not containing a valid DOI name (B). As the DOI names may not be expressed as search term and the broad term doi+10 is used as search term, there are also repositories without a DOI reference gathered. These repositories are discarded by using the Python re library for regular expression. Based on the regular expression for a valid DOI, that matches 74.4M of 74.9M DOIs:

```
/\d{4,9}/[-;._/:A-Za-z0-9]+$/i
```

the following regular expression captures the DOI prefix and suffix:

```
(10\.[0-9]{4,}(?:[.][0-9]+)*\s?/\s?[-;._/:A-Za-z0-9]+)
```

In addition, it also covers DOI references with a blank before and after the forward slash, like in the repository `https://github.com/vtomasv/uiNin` with the DOI name 10.1007 / 978-3-319-00395-5. Some of the extracted DOI names end with a full stop or a closing parenthesis, like `https://doi.org/10.1371/journal.pcbi.1007584`). It cannot be determined whether these characters are part of the DOI name, like in `https://doi.org/10.1061/(ASCE)0733-9437(1993)119:2(334)`, so they are not cutted off. This is handled during the collection of the publication metadata by requesting the DOI name.

To facilitate the application of the long DOI names, the International DOI Foundation provides a public service, named shortDOI Service, to create aliases for DOI names.²¹

3. Analysis of Research Software

As example, the shortDOI Service states for https://dx.doi.org/10.1002/(SICI)1097-0258(19980815/30)17:15/16%3C1661::AID-SIM968%3E3.0.CO;2-2 the shortcut 10/aabbe and the shortcut HTTP URI https://doi.org/aabbe. There should be no difference between the shortDOI and the DOI for applications that resolve DOI names. Such a shortDOI link, like https://doi.org/10/gf4wzh, is used in the repository https://github.com/aaronspring/Spring_and_Ilyina_2019_code. The shortDOI 10/gf4wzh can be resolved by Crossref. As a result, there are three more valid DOI forms, that have to be captured. To extract these shortDOIs, an individual regular expression is required:

```regex
((?:doi\.org|\s10|doi\.org/10|doi\[\s\]?10)/\[a-zA-Z0-9\]+)\['\s><)\]
```

These two regular expressions are applied to the description and Readme file of each repository, see Figure 3.20. The new database entry, consisting of the repository’s full name, its hosting service, and its list of DOI names, is inserted to the the rsRepositories database table. As preparation for the analysis, the group name is also added to the database entry. It relates to the harvesting source and may be set to github, acm, or arxiv. In the case of an existing repository entry, the DOIs are appended to its references list. Besides the database table for research software repositories, a second database table for research software publications is built, named rsPublications. Each detected DOI with its associated repository name is inserted. If an entry for a DOI already exists, the repository name is added to its repository list.

This procedure has to iterate over all harvested repositories in the Repositories database table. Since there is no anchor to determine the last repository before interrupting the iteration, this step should be executed without a break. Nevertheless, should an error occur or there arises another reason for a restart, the process may be started again. Because it is checked whether an entry for the repository already exists, before inserting a new document to the database table.

Repositories Referenced in Publications

Included in the harvested metadata of the publications is the context of the search term (C). The names of the referenced repositories hosted on GitHub are extracted with the following regular expressions:

```regex
(?i)github\.(com/)?\{[a-z0-9][a-z0-9-]*\([a-z0-9-]*/[a-z0-9-_.-]+\)+
```

The regular expression extracts the repository name in a github.(com/{owner}/{repository}. In accordance with the directives of GitHub, a valid owner name starts with an alphanumeric character. There exist also owner names consisting of one single alphanumeric character. Therefore, this starting character may be followed by alphanumeric characters and single hyphens. It is not allowed that the name ends with a hyphen. Considering this fact would complicate the formulation of an appropriate regular expression and is therefore excluded.
3.6. Research Software Identifier

Figure 3.20. The Sequence Diagram for the DOI extraction of the Research Software Identifier Jupyter Notebook
3. Analysis of Research Software

The owners name is separated from the repository’s name by a forward slash. When extracting the owner, the github.com/owner link has to end before or after the forward slash, as the repositories are of main interest and not the owners in github.com/owner/repository links. The repository name may consist of alphanumeric characters in combination with full stops, underscores, and hyphens. The combination of the owners and repository’s name has to be unique.

The repositories, referenced in research publications, complement the rs_repositories database table. Within the harvested publications are also publications that only contain a link to GitHub, https://github.com. These publications are not relevant for the analysis and discarded from further processing. The extracted repository names are looked up in the rs_repositories database table. If an entry for a repository exists, the publication is added to its reference list, provided that it is not already included. If a new entry for a repository is required, its full name and the current reference list are inserted. Like in the previous step, the rs_artifacts database table is filled. If a publication only references the URL github.com, it is not inserted. The preferred identifier of a publication is a DOI name. If it is not available, the arxiv identifier is registered.

If an interruption occurs, this cell can be started again. It is iterated over the whole publications database table, but the search for existing entries in the database table avoids double entries for one repository.

GitHub Owner

Most commonly the GitHub URL in a publication references a repository, but there are also GitHub repository owners linked with URLs in the form of github.com/owner (D). These user names are looked up in the text fragments of each harvested publication with the regular expression:

```
(?i)github\.com/ *(a-z0-9[a-z0-9-]*)/\?(?:s|\.|\)|\'|\"|$|\;|\}|,)
```

It is based on the definition of valid owner names, as described in Section 3.6.2. Before requesting the repositories for each owner, the name is verified against a list of GitHub sites that meet the criteria of a valid owner name. These include explore, topics, trending, collections, events, and features. The repositories are gathered immediately after looking up existing GitHub owner links in a publication. In doing so, the publication metadata are immediately assigned to the new repository entry in the rs_repositories database table. For the requests the general harvester method of the of the GitHubHarvester is used, described in Section 3.5.3 and illustrated in Figure 3.18.

In the case of no existing repositories, the owner is not considered any further. Otherwise, the rs_repositories and rs_publications database tables are updated. For existing repository entries, the publication is added to their references list and the new repository is appended to the repositories list of existing publication entries. If an entry for a repository
or a publication does not exist yet, the a new one is created. Like for the previous GitHub REST API request, the process sleeps for the fixed time.

**Harvesting Repository Information**

For the next three steps (E, F, and G) the GitHubHarvester class is required. Since the repository names are only extracted from the text fragments and inserted into the database table (C), their metadata are missing. For each entry in the rs_repositories database table the corresponding metadata are looked up in the repositories database table (E). Absent metadata are requested with the general harvester method of the GitHubHarvester class, described in Section 3.5.3 and illustrated in Figure 3.18. When the request is not successful, the suffix of the current repository name is checked with an auxiliary function. If the last character is not alphanumeric or the name ends with one of the following suffixes, the character respectively the suffix are cut off and the metadata are requested again.

```
['.git', '.The', 'pdf', '.svg', '.In', 'fulltext', 'meta', '.html', 'abstract',
 'To', 'full', 'status', 'epdf', 'full', 'jsessionid', 'users', 'badges',
 'issuetoc', 'suppinfo']
```

Moreover, duplicated repositories in the rs_database database table are removed. This step is strongly marked by operations on the database tables.

Also the content review process (F) is based on a method in the GitHubHarvester class, that relies on the general harvester class. After requesting the content of a repository, an additional quick check is applied. By iterating over the content, it is verified that the repository includes further files except the README, LICENSE, and .gitignore files. If this does not apply, the repository is removed from the rs_repositories database table.

To harvest the first commit dates of the repositories (G), a GitHubHarvester class method is implemented that is build upon the general harvester method. Initially, the last commit is requested with one commit per page. The HTTP(S) response header contains the link for the first commit, that corresponds to the last page of the pagination through all commits. With a second request the information about the first commit are gathered. The dates of the first and last commit are extracted from the responses and returned to the Notebook process. Based on this dates, the lifespan and the active status are computed and afterwards are inserted to the corresponding database entry.

**DOI Audit**

The DOI metadata of the research software artifacts are required to determine their research subject, and thus of the related research software repositories. By iterating over all artifacts of the rs_artifacts database table that are identified by an DOI, the DOI metadata are requested from the Crossref API. The regular expression does not always extract the correct DOI, which leads to unsuccessful requests. Like for the repository names, the last character and suffix of the DOI are checked with an auxiliary function and shortened
3. Analysis of Research Software

where appropriate. A successful request of metadata confirms the shortened DOI name and it is updated in the rs_artifacts and rs_repositories database tables. Failed requests may also be caused by shortDOIs. In this case, the doi.org API is used to verify whether the DOI has an alias that can be used for another Crossref API request.

Research Subjects

The last step from the repository preparation step is the assignment of a research subject to the repositories. By means of the ISSN and ISBN of the related publications, the research subject is looked up in the publication_subjects database table. It combines the journal name, together with its ISSN, ISBN, and title of the Scopus source list and book title list with the detailed information of the All Science Journal Classification Codes (ASJC) of Scopus.\textsuperscript{22} The ASJC specifies the four research subjects life sciences, social sciences, physical sciences, and health sciences. These are refined into 26 research fields, that are listed in Appendix C. One additional research field is Multidisciplinary, that has no assigned research subject.

Since for the publications with an arXiv ID no ISSN or ISBN is present, the arXiv taxonomy is used to assign a research subject.\textsuperscript{23} The categories are considered as research fields and according to the ASJC a research subject is assigned to this research fields.

3.7 Framework Validation

The framework was designed respecting the best practices for the development of notebooks that are introduced in Section 2.3.4. In the following, the framework is reflected upon the realization of the quality characteristics of the ISO/IEC 25010 and the best practices. Focused are the functionality, usability, maintainability, and portability, since these characteristics are the first-order requirements.

Functionality Initially, all possible research software repository candidates are identified from the harvested repositories and publications. During the data cleansing steps of the Research Software Identifier some of the candidates are discarded. Table 3.2 illustrates the course of the research software repositories and research software artifacts quantity. After each step a random sample of 50 discarded repositories is reviewed to verify the correct operation.

The first step is the extraction of valid DOIs from the 65,857 harvested repositories. 8,200 research software repository candidates are refused as they do not contain a valid DOI name, resulting in 57,657 research software candidates and 106,155 referenced DOIs. After the extraction of the repository names from the gathered publications, the number of research software repository candidates increases to 87,119. By adding the repositories of the referenced GitHub user names, the total number of research software candidates

\textsuperscript{22}https://www.scopus.com
\textsuperscript{23}https://arxiv.org/category_taxonomy
3.7. Framework Validation

Table 3.2. Development of the research software candidates towards research software

<table>
<thead>
<tr>
<th>Identification Step</th>
<th>Research Software Repositories</th>
<th>Research Software Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Quantity</td>
<td>65,857</td>
<td>30,880</td>
</tr>
<tr>
<td>DOI</td>
<td>57,657</td>
<td>106,155</td>
</tr>
<tr>
<td>Repo Name</td>
<td>87,119</td>
<td>131,783</td>
</tr>
<tr>
<td>User Name</td>
<td>210,797</td>
<td>132,955</td>
</tr>
<tr>
<td>Metadata</td>
<td>203,700</td>
<td>132,955</td>
</tr>
<tr>
<td>Content</td>
<td>198,462</td>
<td>132,955</td>
</tr>
<tr>
<td>Commits</td>
<td>198,115</td>
<td>132,955</td>
</tr>
<tr>
<td>Crossref</td>
<td>200,508</td>
<td>111,512</td>
</tr>
</tbody>
</table>

extends to 210,797 repositories. This quantity is reduced to 205,881 repositories, as for 4,916 repository names no metadata could be requested. Among them are 21,648 repositories without a primary language.

106,155 DOI names are gathered from the harvested GitHub repositories. Added are 25,628 of 30,880 for the search term `github.com` harvested publications, as they contain at least one GitHub repository name. So the `rs_publications` database table contains 131,783 research software publication candidates. Among them, 2,308 publications contain a repository owner.

For the validation of the overall Notebook, a random sample of 100 repositories was created. This sample was manually reviewed resulting in a classification of the repositories in research software repositories and no research software repository. After the data processing, it is checked whether the identified research software repositories are in the `rs_repositories` database table and whether the others are not contained. All repositories in the random sample are classified correctly by the framework.

**Usability** All Notebooks have short titles, like `publication_harvester` or `rs_identifier`, and a markdown heading for the description (principle 1). Also within the Notebook, markdown cells describe the functionality of the following cell (principle 2). Since the harvester classes are defined as Python modules, the harvester Notebooks are clearly arranged (principle 3). The Research Software Identifier and Research Software Analyzer consists of a number of small steps. To increase the transparency, these Notebooks may be divided into smaller Notebooks, like an individual Notebook for the data cleansing and the data preparation and one Notebook for each research question.

**Maintainability** The framework has a modular structure (principle 3). All modules obtain their relevant data from a central database (principle 7). Thus, individual modules can be changed or modified without considerable effort. This is also ensured by a loose coupling of the modules. The modular design permits the extensibility and modifiability of this
3. Analysis of Research Software

application. This characteristic is also promoted within the structure of the modules. By means of the configuration file individual processing steps can be selected.

**Portability** All required packages are imported at the beginning of each Notebook (principle 6) and a `requirements.txt` is provided (principle 4). Moreover, the release with Binder ensures the platform independence. More details are provided in Chapter 4.
Chapter 4

Set Up a Research Software Observatory

For the implementation of the Research Software Observatory the container approach is chosen, as described in Section 2.3.5. This concept requires an execution environment for the software, as well as metadata for the classification and citeability. To comply with the FAIR and Open Science principles, the recommendations of Hasselbring et al. [2019] is followed.

As illustrated in Figure 4.1, a public GitHub repository for the framework is set up. Besides the source code, it contains a requirements.txt file, license information, a CITATION.cff file, a codemeta.json file, and a folder with files for Binder. These are required to set up a Binder repository, where the framework can be tested and used for smaller-scale projects. Binder builds and registers a Docker image from the GitHub repository. Afterwards, it connects with JupyterHub and provides a public IP address for the interaction with the framework.

To permanently store the current state of the framework, the general-purpose open-access repository Zenodo is chosen. Also the MongoDB database dump is archived on Zenodo. Since Zenodo has a build in integration for GitHub, the GitHub account can be used to log in. Provided that Zenodo is authorized to connect to the GitHub account, which includes the permission for the use of webhooks on the public repositories, a non-standardized procedure for bilateral server communication. If the webhook is set up between the repository and Zenodo, new releases of the repository may be easily archived. A DOI is assigned to each new release and. During the DOI registration process the metadata is sent to servers of Datacite, a not-for-profit organization to promote and improve data citation.

The FAIR Guiding principles are implemented as follows.

**Findability**  The metadata of the framework are specified in the codemeta.json file and the CITATION.cff file.

**Accessibility**  The current version of the framework is archived on Zenodo and is identified by the DOI 10.5281/zenodo.4277955. The MongoDB database dump is also archived on Zenode and is identified by the DOI 10.5281/zenodo.4260761.
4. Set Up a Research Software Observatory

**Interoperability** Interfaces are provided for the harvester classes. Also the used database can be replaced by changing the `_Database` class in the database module. Each Notebook may be used independently, except the Study Reproducer.

**Reusability** For smaller-scale projects and testing purposes the Binder repository is provided. The `requirements.txt` file contains all dependencies, so that the Notebooks may be executed in a Python environment.

![Deployment Diagram of the framework](image-url)
Chapter 5

Evaluation

5.1 Description of the Data Sets

5.1.1 Repositories Set

In the present study, 211,860 repository metadata are harvested according to the specified search terms doi+10 and doi+10+in:readme. These repositories are only a small part of more than 62 million public repositories hosted on GitHub.\(^1\) Figure 5.1 surveys the increasing number of public repositories by creation date, the first repository has been established in 2007.\(^2\) Among the total, there are only a few (2.23%) archived repositories, as shown in Table 5.1. Nearly a half (43.07%) of the repositories is forked, whereas not many repositories (10.09%) are forks. In the metadata of half of the repositories (51.75%) is stated that they have a license. And nearly all repositories (89.77%) have an assigned language.

Additionally, it has to be explained that 65,857 repositories are directly requested from GitHub, almost a third (30.89 %) have a Readme file. In return, 417 (0.64%) repositories do not have a Readme file. For the majority (68.92%) the Readme file is not requested.

### Table 5.1. Repositories set overview

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Percentage</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall sample</td>
<td>100%</td>
<td>211,860</td>
</tr>
<tr>
<td>Available license</td>
<td>51.75%</td>
<td>109,592</td>
</tr>
<tr>
<td>Assigned language</td>
<td>89.77%</td>
<td>190,193</td>
</tr>
<tr>
<td>Forked repositories</td>
<td>43.07%</td>
<td>91,254</td>
</tr>
<tr>
<td>Forks</td>
<td>10.09%</td>
<td>21,379</td>
</tr>
<tr>
<td>Archived repositories</td>
<td>2.23 %</td>
<td>4,732</td>
</tr>
</tbody>
</table>

\(^1\) requested on November 12, 2020 with https://api.github.com/search/repositories?q=is:public&per_page=1

\(^2\) requested on November 12, 2020 with 'https://api.github.com/search/repositories?q=is:public+created:YYYY&per_page=1'
5. Evaluation

![Chart showing growth of public repositories hosted on GitHub from 2007 to 2020]

Figure 5.1. Growth of public repositories hosted on GitHub

5.1.2 Publications Set

In total, 30,880 publications are harvested, as shown in Table 5.2. Three quarters of the publications (73.92%) originate from ACM and the remaining publications (26.08%) from arXiv. The DOI is the preferred identifier for the publications. It is provided for most of the publications (69.04%). Very few publications (0.03%) are unnamed, nevertheless they are recognized by a DOI. These articles are all published by ACM, namely five articles in the journal Computer Music Journal, Volume 43, Issue 2-3, four articles in Bioinformatics, Volume 27 (2011), 28 (2012), 29 (2013), and one article in SIGMOD ’14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Percentage</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall sample</td>
<td>100%</td>
<td>30,880</td>
</tr>
<tr>
<td>ACM</td>
<td>73.92%</td>
<td>22,825</td>
</tr>
<tr>
<td>arXiv</td>
<td>26.08%</td>
<td>8,055</td>
</tr>
<tr>
<td>DOI</td>
<td>69.04%</td>
<td>21,320</td>
</tr>
<tr>
<td>No title</td>
<td>0.03%</td>
<td>10</td>
</tr>
<tr>
<td>No published date</td>
<td>0.49%</td>
<td>151</td>
</tr>
</tbody>
</table>
5.1. Description of the Data Sets

Table 5.3. Overview of the metadata sample, the DOI sample, and the overall sample

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Metadata</th>
<th>DOI</th>
<th>Overall</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOI</td>
<td>100%</td>
<td>100%</td>
<td>94.42%</td>
<td>105,287</td>
</tr>
<tr>
<td>DOI metadata</td>
<td>100%</td>
<td>83.21%</td>
<td>78.56%</td>
<td>87,608</td>
</tr>
<tr>
<td>Type specification</td>
<td>99.99%</td>
<td>83.2%</td>
<td>78.56%</td>
<td>87,608</td>
</tr>
<tr>
<td>ISSN or ISBN</td>
<td>94.44%</td>
<td>78.58%</td>
<td>74.19%</td>
<td>82,735</td>
</tr>
<tr>
<td>ISSN</td>
<td>73.93%</td>
<td>61.52%</td>
<td>58.08%</td>
<td>64,771</td>
</tr>
<tr>
<td>ISBN</td>
<td>23.5%</td>
<td>19.55%</td>
<td>18.46%</td>
<td>20,584</td>
</tr>
<tr>
<td>Main subject</td>
<td>64.85%</td>
<td>53.96%</td>
<td>50.95%</td>
<td>56,811</td>
</tr>
</tbody>
</table>

(a) Artifacts in the DOI subsample

(b) Artifacts with harvested metadata

Figure 5.2. Distribution of the type categories for artifacts

5.1.3 Software Artifacts Set

The total number of research software artifacts, in the following referred to as artifacts, sums up to 111,512. The majority of the artifacts (94.42%) is identified by a DOI and the remaining (5.58%) by an arXiv ID. For the subsample that comprises artifacts identified by a DOI, in the following DOI sample, the metadata are gathered. The requests are successful for most of the artifacts (83.21 %), as shown in Table 5.3.

The artifacts with requested DOI metadata are considered in more detail, in the following referred to as metadata sample. The type specification is one of the available informations. Only for seven artifacts, the type is not set. Added to the number of artifacts with missing metadata, it amounts to a share of 16.8% artifacts in the DOI sample. The majority of the artifacts is classified as journal article, followed by proceedings articles, as shown in Figure 5.2b. Figure 5.2a illustrates the type distribution in the DOI sample. The aggregated category “Others” summarizes – among other types – books, data sets, journals, and monographs.

Nearly all artifacts (94.44 %) with harvested metadata have an assigned ISSN or ISBN,
5. Evaluation

![Diagram](image)

**Figure 5.3.** Numbers of artifacts with assigned characteristics, especially ISSN and ISBN

as shown in Table 5.3. This corresponds to 78.58 % in the DOI subsample. Figure 5.3 illustrates the relation between the DOI, metadata, ISSN, ISBN, and subject characteristics. For a few artifacts both ISSN and ISBN are available (2.99 % and 2.49% respectively; total 2,620 in the DOI sample). This explains the difference between the value for artifacts with ISSN or ISBN (82,735) and the summed value for artifacts with ISSN and artifacts with ISBN (64,771 + 20,584 = 85,355). On the basis of the ISSN and ISBN, the main subject of an artifact, and thus of a research software repository, is derived. An assignment is obtained for more than a half of the artifacts. And nearly all journal articles have an allocated main subject (92.37% and 90.75% respectively in the DOI sample).

The subject of an artifact is a further information that is available in metadata. One third of the artifacts have a subject category in their DOI metadata. This category corresponds to the subcategory of the subject assigned by the framework. Both subject informations are available for 28.51% of the research artifacts. The subject of the DOI metadata differs from the subcategory only for a few artifacts (6.6%). In the majority of the cases (42.73%) the DOI subject is ['General Biochemistry, Genetics and Molecular Biology’, ‘General Agricultural and Biological Sciences’, ‘General Medicine’] and the subcategory of the framework subject is ['Multidisciplinary’].

5.1.4 Research Software Repositories Set

In total, 211,860 repositories are harvested, of which 198,115 are selected for the research software repository set. Most of the research software repositories, in the following referred to as repositories, are set up in recent years, as illustrated in Figure 5.4a. It has to be mentioned that the year 2020 contains all repositories until the harvesting date, namely
October 12, 2020. The oldest repositories have their first commit date in the 1970s, in total 29 repositories. Here, it is important to discriminate the creation time of a repository from the date of its first commit. For a request regarding repositories created in 1970, the answer is zero. For example, the repository dspinellis/unix-history-repo had been created on July 18, 2014, but the first commit was captured on January 1, 1970.

Each repository is characterized by its reference type. The reference allows the allocation of a repository into one of three groups. Thus, the GitHub group consists of repositories that are gathered from GitHub and contain a valid DOI reference. Repositories referenced in an ACM publication, are part of the ACM group. The arXiv group comprises all repositories that are linked in arXiv publications. Figure 5.5a shows that the majority of the repositories is referenced by ACM publications. A large part within the ACM group are the gitpan repositories, in total 36,320. Their membership is caused by the link github.com/gitpan in an ACM article. Special attention claims the GitHub group, because it is the analysis group providing most main subjects Figure 5.5b.

The majority of the repositories belongs to one of the three analysis groups, as shown in Table 5.4. Only a few repositories are referenced in both an ACM and arXiv publication and contain a DOI reference. When there exists a mutual linking, than most commonly between GitHub repositories and ACM publications, as shown in Figure 5.6a.

A decisive feature for research software is the research publication. Whether a repository

---

5 requested on November 12, 2020 with https://api.github.com/search/repositories?q=created:1970&per_page=1
5. Evaluation

<table>
<thead>
<tr>
<th>Sample</th>
<th>GitHub</th>
<th>ACM</th>
<th>arXiv</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall sample</strong></td>
<td>28.4% (56,255)</td>
<td>68.36% (135,422)</td>
<td>4.67% (9,254)</td>
</tr>
<tr>
<td><strong>Repositories with main subject</strong></td>
<td>54.12% (38,324)</td>
<td>35.64% (25,240)</td>
<td>12.63% (8,941)</td>
</tr>
</tbody>
</table>

![Graph](image1.png)  
**Figure 5.5.** The number of repositories in the individual analysis groups

![Graph](image2.png)  
**Figure 5.6.** The number of repositories belonging to pairwise analysis groups
has a linked research publication, is determined by a present subject. It is assigned to the repository via its associated artifacts. If an ISSN or ISBN is available and a subject can be allocated, it is assumed, that the artifact is a research publication. Therefore, the research software repositories with a subject are analyzed more closely.

Like in the overall sample, the majority of the repositories is set up in the last years. With the strong growth of the repository quantity in general from 2010 onwards, also the number of research software repositories raises. The number of repositories with a verified associated publication remains less than the repositories containing a DOI. Also the growth of the number started nearly a decade later.

The membership in analysis groups shows a similar situation as for the overall sample. Most of the repositories belong to one analysis group, as shown in Table 5.4. Figure 5.6b illustrates that the mutual reference of a repository and a research publication apply to only a few repositories. Also the repository reference in research publications from both ACM and arXiv is not common. The group sizes regarding the repositories with a main subject differ from the ones for the overall sample. Figure 5.5b illustrates that GitHub is the largest group, followed by ACM and arXiv.

5.2 RQ1: Research Subjects

According to the All Science Journal Classification (ASJC) of Scopus, the four research subjects are health sciences, life sciences, physical sciences, and social sciences. They are refined by 26 research fields. The general research field Multidisciplinary does not possess any assigned research subject. For the following analysis, it is assumed that the research subject of Multidisciplinary is also Multidisciplinary. Besides this supplement, the additional category Mixed is introduced to represent interdisciplinary repositories, namely repositories with more than one research subject. This is due, when more than one publication is related or the related publication is multidisciplinary. Juxtaposed is the analysis of the research subjects where the Mixed category is separated into its components and each component feeds into the analysis.

In the following, results of analyses will explain the spreading of research subjects in different analysis groups. The groups are named the GitHub Sample, the ACM Sample, the arXiv Sample and the Overall Sample. The latter puts the three groups together.

Overall Sample

Initially, the research subjects of the overall sample are regarded. Analysis of the sampled data is illustrated in Figure 5.7a, which shows that the majority of these repositories belongs to physical sciences, followed by repositories with multiple subjects. If one considers all research subjects of the Mixed category separately, the number of assigned repositories increases in life and health sciences, as shown in Figure 5.7b.
5. Evaluation

Figure 5.7. Research subjects of the research software repositories, all three analysis groups combined: GitHub–ACM–arXiv

Figure 5.8 presents the research fields of the repositories to make a research subject refinement. With up to 27.1% of the repositories, computer science is the most strongly represented individual research field. But the majority of repositories (55.3%) is referenced by publications, respectively they refer to publications, from multiple research fields. The remaining repositories pertain to various research fields, like biochemistry, genetics and molecular biology, earth and planetary sciences, and further research fields accumulated in the Others category.

Figure 5.9 presents the distribution of repositories regarding also the individual research fields from the Mixed category in the preceding Figure. Overall, there is a large variation across the 44 research fields. Repositories with a share of less than 2.0% are accumulated to 10.8% in the category Others. Among them are business, management and accounting, veterinary, and dentistry. All fields, together with their quantity of research software repositories, are presented in Appendix C.

GitHub Sample

A closer look into the GitHub sample, shown in Figure 5.10, detects that in physical sciences the repositories are most commonly referenced, followed by the life and health sciences. These three subjects also often refer to the same repositories, since the Mixed category has a large share (39.2%). Counting each subject in the Mixed category separately, enhances the sector of the life sciences remarkably (25.8%), followed by the physical and health sciences.

The majority of the repositories in the GitHub group references such publications that are assigned to multiple research fields. In the other case, the repository refers to more than one publication belonging to different research fields. This is expressed in the
5.2. RQ1: Research Subjects

![Pie chart showing the distribution of research subjects.]

**Figure 5.8.** Named research fields of the research software repositories. Overall Sample with Mixed category.

![Pie chart showing the distribution of research fields.]

**Figure 5.9.** Research fields of the research software repositories. Previously uniform Mixed category in Overall Sample is specified.

A large proportion of the Mixed category that occupies 65.7% in Figure 5.11. The remaining references are related to research publications from various research fields. Breaking down the aforementioned Mixed category in Figure 5.12, depicts that publications belonging to computer science (21.7%) and biochemistry together with genetics and molecular biology (14.1%) are slightly more commonly referenced than publications of other research fields.
5. Evaluation

![Graph showing research subjects and categories.](image)

**Figure 5.10.** Research subjects of the GitHub research software repositories

**Figure 5.11.** Research fields of the research software repositories in the GitHub group. The category Mixed dominates with 65.7%

**ACM Sample**

Most of the repositories referenced by ACM publications are assigned to physical sciences (80.9%), as presented in Figure 5.13a. The percentage does not dramatically change, when the interdisciplinary repositories become split in Figure 5.13b. Then the physical sciences make 85.9% with great distance to social sciences and life sciences (6.3% and 5.7%, respectively).

Regarding the research fields, Figure 5.14 depicts that most of the repositories (53.4%)
5.2. RQ1: Research Subjects

Agricultural and Biological Sciences
Biochemistry, Genetics and Molecular Biology
Chemistry
Medicine
Multidisciplinary
Computer Science
Environmental Science
Physics and Astronomy
Earth and Planetary Sciences
Mathematics
Engineering
Decision Sciences
Others

Figure 5.12. Research fields of the research software repositories in the GitHub group

(a) Sample with Mixed category
(b) Mixed category splitted

Figure 5.13. Research subjects of the research software repositories referenced in ACM publications are referenced by interdisciplinary or multiple ACM publications, followed at 43.9% by computer science publications. The interdisciplinary field, represented by the Mixed category, consists mainly of repositories belonging to computer science, mathematics (15.0%), and not strongly represented research fields, since the share of the Others category increases from 1.2% to 16.2%.

arXiv Sample

Surpassing the ACM sample, the majority of the repositories is referenced by arXiv publications assigned to physical sciences (95.0%), as shown in Figure 5.15a. The interdisciplinary
5. Evaluation

Figure 5.14. Research fields for the research software repositories referenced in ACM publications

Figure 5.15. Research subjects for the research software repositories referenced in arXiv publications

repositories of the Mixed category belong mainly to the life sciences (2.8% only) and evenly distributed to the other subjects (< 1%), as illustrated in Figure 5.15b.

The metadata of arXiv publications provide information about the primary research fields. Figure 5.16a depicts that 64.6% of the sampled publications belong to computer science and most commonly reference a GitHub repository, followed by publications from earth and planetary sciences (7.7%). The remaining quarter of repositories is related to publications from various fields.

Regarding the computer science field (Figure 5.16b), the majority of repositories is assigned to computer vision and pattern recognition (41.8%). A portion of 20.1% is devoted to machine learning. Also publications belonging to robotics (12.4%) and to computation and language (11.7%) refer to GitHub repositories, while the remaining quarter of repositories is referenced from publications of various other disciplines in computer science.
5.3 RQ2: Research Software Sustainability

The lifespan of a repository is computed by means of the first and the last commit. If the last commit occurred in the past 12 months, the repository is considered active. Table 5.6 shows that two active and eleven dormant repositories of the overall sample have a negative lifespan. For instance, the timestamp for the first commit of the active repository MrHeadShok/bioinfo–JS is set to May 13rd, 2020 (01:55:24) and the last commit is captured on May 13rd, 2020 (01:00:46). A larger difference between the first (2019-06-14, 15:55:10) and last (2016-09-07, 18:04:02) commit, namely -1010 days, is observed for the repository microsoft/CoSConfiguredProjectSample.

The most long lasting, active repository is angular/angular.js with a lifetime of 18,559 days (50y 9m). Its first commit is captured on January 1st, 1970. By far most repositories with a lifespan greater than 12 years belong to the ACM group, while in the arXiv group exceedingly few.

In the following analysis are the two additional samples computer science and computational science included. The computer science sample comprises all repositories that are assigned to the research field computer science. Interdisciplinary repositories are also captured. The remaining repositories, not assigned to computer science, are accumulated in the computational science sample. In the overall sample nearly half of the repositories (44.35%) are live, as presented in Table 5.7. This also applies for the repositories in the GitHub sample (49.59%), as illustrated in Figure 5.17, and the repositories that are assigned to computational science (48.71%). Figure 5.18 depicts that the least number of live repositories (44.35%) are referenced by ACM publications, followed by the repositories (41.64%) that belong to computer science. The largest share of live repositories (59.76%) is pooled in the arXiv sample, as presented in Figure 5.19. This sample also has the lowest proportion
5. Evaluation

Table 5.6. Number of repositories with a negative lifespan or a lifespan greater than 12 years

<table>
<thead>
<tr>
<th>Sample</th>
<th>Lifespan &lt; 0 Years</th>
<th>Lifespan &gt; 12 Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall active sample</td>
<td>2</td>
<td>262</td>
</tr>
<tr>
<td>Overall dormant sample</td>
<td>11</td>
<td>179</td>
</tr>
<tr>
<td>GitHub active sample</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>GitHub dormant sample</td>
<td>2</td>
<td>51</td>
</tr>
<tr>
<td>arXiv active sample</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>arXiv dormant sample</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>ACM active sample</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>ACM dormant sample</td>
<td>9</td>
<td>120</td>
</tr>
</tbody>
</table>

Figure 5.17. Lifespan of the repositories referencing a research publication, limited to 1 year

of repositories that have a lifespan of one day (8.03%). All other samples have a share of this short-lived repositories in the range of 12.23% to 16.84%.

Most of the repositories that are referenced by arXiv publications are currently live, but their lifespan distribution has the lowest median with 203 days (seven months). The repositories of the GitHub sample (214 days) and the computational science sample (212 days) have a barely higher median. Appreciably higher is the median of the computer science sample (292 days) and the ACM sample (399 days). If only the live repositories are considered, the median of the ACM sample is doubled (832 days). For the GitHub sample the median increases to 369 days, while the median of the arXiv sample has increased only slightly to 226 days.
5.4 RQ3: Categories of Research Software

Table 5.7. The number of repositories regarding the characteristics live, dormant, and live for one day

<table>
<thead>
<tr>
<th>Sample</th>
<th># Repos</th>
<th># Life</th>
<th># Dormant</th>
<th># 1 day life</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>70,706</td>
<td>31,355</td>
<td>39,351</td>
<td>10,162</td>
<td>258 days</td>
</tr>
<tr>
<td></td>
<td>(44.35%)</td>
<td>(55.65%)</td>
<td>(55.65%)</td>
<td>(14.37%)</td>
<td></td>
</tr>
<tr>
<td>GitHub</td>
<td>38,257</td>
<td>18,970</td>
<td>19,287</td>
<td>6,442</td>
<td>214 days</td>
</tr>
<tr>
<td></td>
<td>(49.59%)</td>
<td>(50.41%)</td>
<td>(50.41%)</td>
<td>(16.84%)</td>
<td></td>
</tr>
<tr>
<td>ACM</td>
<td>25,221</td>
<td>8,011</td>
<td>17,210</td>
<td>3,085</td>
<td>399 days</td>
</tr>
<tr>
<td></td>
<td>(31.76%)</td>
<td>(68.24%)</td>
<td>(68.24%)</td>
<td>(12.23%)</td>
<td></td>
</tr>
<tr>
<td>arXiv</td>
<td>8,920</td>
<td>5,331</td>
<td>3,589</td>
<td>716</td>
<td>203 days</td>
</tr>
<tr>
<td></td>
<td>(59.76%)</td>
<td>(59.76%)</td>
<td>(59.76%)</td>
<td>(8.03%)</td>
<td></td>
</tr>
<tr>
<td>Computer Science</td>
<td>43,648</td>
<td>18,174</td>
<td>25,474</td>
<td>5,890</td>
<td>292 days</td>
</tr>
<tr>
<td></td>
<td>(41.64%)</td>
<td>(58.36%)</td>
<td>(58.36%)</td>
<td>(13.49%)</td>
<td></td>
</tr>
<tr>
<td>Computational Science</td>
<td>27,058</td>
<td>13,181</td>
<td>13,877</td>
<td>4,272</td>
<td>212 days</td>
</tr>
<tr>
<td></td>
<td>(48.71%)</td>
<td>(51.29%)</td>
<td>(51.29%)</td>
<td>(15.79%)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.18. Lifespan of the repositories referenced in ACM publications, limited to 1 year

5.4 RQ3: Categories of Research Software

In total, 135,422 repositories are referenced in ACM publications. A subject is assigned to 25,240 repositories. To determine the most valuable repositories, the average of the received stars, the number of watchers, and the number of forks is computed. The 30 most valuable
5. Evaluation

repositories are qualitatively evaluated regarding their category and their relation to the publication. Table 5.8 illustrates the top ten repositories.

Most of the evaluated repositories are known and frequently used open-source frameworks or tools, like Angular, and OpenCV. Particularly well represented are frameworks from the area of artificial intelligence, like Tensorflow, Keras, Caffe and Transformers. Also the code editor Atom and programming languages are represented, like Swift, Ruby on Rails, and Julia, a high-level, high-performance dynamic language for technical computing. A lot of the repositories are cited as an example or used as test cases for other frameworks that deal, for example, with graph analysis. Eight of these repositories are subject in one paper that addressed the challenges of open-sourcing proprietary software projects. Since the repositories are all actively maintained, no representative for the abandoned research software could be found. The Caffe framework represents the actively maintained output of research, as the project was developed during a PHD. However, in the present publications it is referenced as research tool.

5.5 Discussion

The public treatment of research software, especially its sustainability, is put into the focus of the present essay. Sustainability is understood as the passable way to persist probably and to be used further by other researchers. This objective comes within reach and effectiveness when software is maintained over a longer period. The duty was to investigate the current sustainability state of research software that is assigned to computer science and computational science.

4https://github.com/JuliaLang/julia
Table 5.8. Repository lifespan overview in days (months, years)

<table>
<thead>
<tr>
<th>Acknowledgment</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>218,880</td>
<td>freeCodeCamp/freeCodeCamp</td>
</tr>
<tr>
<td>127,205</td>
<td>tensorflow/tensorflow</td>
</tr>
<tr>
<td>124,938</td>
<td>vuejs/vue</td>
</tr>
<tr>
<td>120,083</td>
<td>twbs/bootstrap</td>
</tr>
<tr>
<td>92,170</td>
<td>github/gitignore</td>
</tr>
<tr>
<td>70,156</td>
<td>d3/d3</td>
</tr>
<tr>
<td>51,161</td>
<td>mrdoob/three.js</td>
</tr>
<tr>
<td>49,192</td>
<td>angular/angular.js</td>
</tr>
<tr>
<td>47,842</td>
<td>microsoft/terminal</td>
</tr>
<tr>
<td>46,234</td>
<td>opencv/opencv</td>
</tr>
</tbody>
</table>

5.5.1 RQ1: Research Subjects

Not quite the half of all repositories are assigned to computer science, whereas the share reduces to nearly one third when regarding the individual research field. This large proportion is based on the repositories that are referenced by ACM and arXiv publications. In contrast, computer science is the strongest represented research field for repositories referencing research publications, but it is only a fifth part. These repositories are spread across various research fields of computational science. The present observations correspond to the findings of the pilot study performed by Hasselbring et al. [2019, 2020b].

The pilot study revealed that most of the publications that are referenced in GitHub repositories belong to computational science, especially to the life sciences, as illustrated in Figure 5.20a. Also in the current distribution of research subjects respectively research findings, the share of the life sciences is similar strong. And the main part of the repositories belongs to research fields of computational science. The number of repositories assigned to computer science has increased in the period between the both studies, but still clearly lags behind the quantity of computational science repositories.

Figure 5.23 illustrates that repositories are most commonly referenced by ACM publications of the research field computer science, especially software engineering, information systems, and social and professional topics. The major proportion of computer science is also indicated in the current study, but the research fields are not broken down to a more detailed view.

Also arXiv publications that reference a GitHub repository are assigned mainly to computer science, followed by astrophysics, as shown in Figure 5.24. Even though, astrophysics is not so strongly represented in the current findings and the distribution of the computational science research fields has modified, the large share of computer science is confirmed. Also the prevalence of artificial topics, as shown in Figure 5.21, namely computer vision and pattern recognition, machine learning, and computation and language, is reflected in
5. Evaluation

Figure 5.20. Research fields of repositories referencing publications or referenced by ACM publications. Data from pilot study by Hasselbring et al. [2019, 2020b]

Figure 5.16b.

### 5.5.2 RQ2: Research Software Sustainability

The repositories referenced by ACM publications are strongly marked by short-lived, dormant repositories. But if the repositories actively maintained, they have a quite long lifespan. The mean value does not level with the mean value of five years in the pilot study, but is the highest mean value of the current samples. Figure 5.23 contrasts the lifespan distributions of the current and pilot study. The difference may be caused the sharp increase of public repositories. In the period from January 2018 to December 2019 about 31 million public repositories are created. If only repositories considered that are created before 2019, the distribution converges to the one from the pilot study, as illustrated in Figure 5.25. In this case, the median of the lifespan increases to nearly five years.

The lifespan of the repositories referenced by arXiv articles is distributed with a median of nearly six months and is only slightly below the median of seven months in the pilot study. This is also reflected in Figure 5.24. Regarding the share of the active repositories, the share decreased from 75% in the pilot study to 60%. But in the current study, the arXiv sample has the strongest proportion of active repositories.

Like in the pilot study, there is an even split of active and dormant repositories in the GitHub sample. The distribution of the lifespan is very similar to the one of the pilot study.
But the lifespan has noticeably increased, from a median of 15 days in the pilot study to 214 days. This change may also be due to the increasing number of public repositories. Moreover, the share of computer science enlarged to 21.7%.

The pilot study revealed that the lifespan of computer science repositories is higher than the lifespan of computational science repositories. Therefore, the repositories are analyzed regarding their assigned research field, namely computer science and computational science. Also for this grouping the mean value of the computer science repositories exceeds the the mean value of the computational science repositories by nearly three months.

5.5.3 RQ3: Categories of Research Software

Overall, the categories and relationships that are represented in the pilot study are identified for the current subsample. The majority of the repositories was used as research tool or was referenced as an example. For further analysis, the selection of the sample has to be reconsidered, since in the current sample are mostly well-known and widely-used frameworks and tools. The evaluation was in particular hampered by inaccessible publications. For instance, the repository d3/d3 has six assigned publications, but not one of them is openly available. Moreover, there are repositories that had not really a relation to the related publication, since only the repository owner is referenced.
5. Evaluation

As Figure 2(a) shows, the computer science software repositories' life spans are distributed with a median of five years. Our hypothesis is that, in computer science research, commercial open source software frameworks are often employed. They are maintained through long time spans by employees of the associated companies.

According to Figure 2(b), the computational science software repositories' life spans have a distribution with a median life span of 15 days. One-third of these repositories remain live for under one day. We hypothesize that, in computational science research, research software is often published only when a corresponding paper has been distributed. The software is not further maintained on GitHub but at some private place, as before (if it is kept up at all).

Examining Figure 2(c), we see that the arXiv repositories are somewhere in between, with a median life span of eight months. Furthermore, 75% of the arXiv repositories are live. We conjecture that in parts of the AI community, the attitude of publishing as early as possible motivates researchers to develop their investigative software openly from the beginning of projects.

Relationships and categories of research software
In addition to the life span, it is interesting to take a closer look at activity related to repositories, such as the number of commits per time unit. We observe different categories and relationships between research publications and research software, including:

- software as an output of research, collaboratively constructed and maintained through an active open source community
- software as an output of research, privately developed but published openly and abandoned after dissemination
- software itself as an object of study or analysis
- software that leads to a fork (in GitHub) that is independently developed as a research output and published openly (if successful, it may be fed back into the original project via GitHub pull requests)

Figure 5.22. Lifespan of repositories referencing research publications
Data from pilot study by Hasselbring et al. [2019, 2020b]

Figure 5.23. Lifespan of repositories referenced by ACM publications
Data from pilot study by Hasselbring et al. [2019, 2020b]
5.5. Discussion

![Graph of current lifespan distribution](a) Current lifespan distribution

![Graph of lifespan distribution in the pilot study](b) Lifespan distribution in the pilot study

Figure 5.24. Lifespan of repositories referenced by arXiv publications
Data from pilot study by Hasselbring et al. [2019, 2020b]

![Graph of pilot study: research areas of arXiv publications referencing GitHub repositories](Figure 5.25. Pilot study: research areas of arXiv publications referencing GitHub repositories)
Several studies address the referencing of software. Often their scope is restricted to individual software tools, programming languages, or research subjects. Howison and Bullard [2016] checked a sample of 90,286 software mentions in 59 biological articles, where software was mentioned in a number of ways. Well over a third (37%) of the mentions were formal publication references, complemented by a few references with a name or website of a project in the references list (7%). The remaining mentions were part of the text by providing the project name (31%) or combined an author or company with a location (18%). Regarding availability of the covered software, one finds the main part accessible (79%). But less than half is freely available (47%), or was provided with its source code (32%). About the used version less information are available (28%), and only a few authors allowed modifications and extensions of the source code (20%).

Wattanakriengkrai et al. [2020] investigated the referencing of research publications in GitHub repositories. In total, 20,278 (0.4%) repositories are harvested that matched the searching pattern for research publication references. The pattern comprised, among other terms, arxiv references and DOI names. Merely around 0.4% of the repositories, mostly assigned to machine learning, referred to a research publication. And mutual referencing is extremely rare. But the majority of papers that are referenced are Open Access.

Jupyter Notebooks are a widely adopted tool in computational science and address central challenges in the data analysis [Rule et al. 2018]. The construction and sharing of Jupyter Notebooks is affected by the characteristics of exploration and explanation. Cluttered Jupyter Notebooks with duplicate cells and similar code snippets are often the result of an exploratory process. Before sharing, these Notebooks require a laborious cleansing and documentation process to perform their explanatory task. Overall, there is a backlog regarding the explanation in Jupyter Notebooks.

Rule et al. 2018 also discovered that the reuse of Jupyter Notebooks is caused by tracking provenance, code reuse, reproducibility of experiments, and presentation of results. However, only few Jupyter Notebooks could be successfully replicated or executed [Pimentel et al. 2019]. Moreover, these authors revealed the realization of good practices in form of descriptive filenames, application of abstractions (complex control flows), and literate programming aspects. The bad practices occur in form of missing dependency declarations, untested code, copy and paste code instead of functions, out-of-order cells, and non-executed code cells.

Wofford et al. 2020 discovered the degree to which Jupyter notebooks facilitate data and
6. Related Work

software citation in astronomy publications. The sample was gathered via the Astrophysics Data System (ADS) using the full text search for “Jupyter”, resulting in 897 ADS records. Fewer than half (48%) referenced one or more Jupyter Notebooks. Most of them are findable and accessible. But this does not apply for only 9% with dead links or restricted access. Over half of all Jupyter Notebooks are referenced by a GitHub link (67%). Alternative references are expressed as DOI issued by Zenodo (5%), download links for a zip file or tar ball (5%), project web site links (4%), and links to static web pages created by NBviewer. Also links to other repositories, Bitbucket, Figshare, and MyBinder are represented (15%). 105 of the 392 references may be considered as citation due to the specified metadata. The motive for citation range from data analysis, reproducibility, tutorials to the integration with one or more products developed by the author. So, in this context the realization of the software citation principles is insufficient, as only one fourth meets the requirements of a citation.

To encourage the collaboration of geospace researchers, to simplify the reproduction of results, and to build open existing work, the pilot project Integrated Geoscience Observatory was launched.\(^1\) This online platform aims to be a unified toolset, combining software and associated data, that are created by individual geoscience research communities.\(^2\) A first approach is the REproducible Software ENvironment (Resen) platform, that entirely depends on open-source software [Bhatt et al. 2020]. Resen provides a containerized Jupyterlab interface by using Python and Docker. This combination ensures the sharing and the reproducibility across the operating systems Linux, macOS, and Windows. Isolated data analysis environments are Docker containers, named buckets. Each container includes a pre-built Docker image, so that a variety of commonly used geospace libraries and software packages are preinstalled.\(^3\) In addition, software to simplify the access and download of data from public repositories, that are made available by various geospace data providers, is also in the buckets. So code can be developed using existing community-developed geospace software. The buckets are customized and configured with the help of the graphical user interface of the JupyterLab server, that is opened in the favored internet browser. The number of buckets per user is not restricted. A prerequisite for the use is the additional installation of Python3 and Docker. To use the complete functionality of Resen, the software must be installed on the own computer and the source code is hosted on GitHub.\(^4\) For demonstrating purposes the cloud version with a user authentication system and limited resources can be used. The complete bucket, containing any custom configuration and the analysis code, may be shared via a research data repository.

---

\(^1\)https://ingeo.datatransport.org/home/
\(^2\)https://www.earthcube.org/group/integrated-geoscience-observatory
\(^3\)https://github.com/EarthCubeInGeo/resen-core
\(^4\)https://github.com/EarthCubeInGeo/resen
Chapter 7

Conclusions and Future Work

7.1 Conclusions

[Hasselbring et al. 2020b] revealed in their pilot study that research software repositories hosted on GitHub mainly reference research publications that are assigned to computational science. Whereas in computer science the research software repositories most commonly are referenced in research publications. Repositories containing a DOI are live as often as dormant, as well as the repositories referenced by ACM publications. The main finding revealed that computer science repositories have a higher lifespan than computational science repositories. Even though the exact number are not obtained, with the developed framework, it is shown that computer science repositories have a noticeably higher lifespan than computational science repositories. Also the general tendencies are revealed. But the strong increase of the dormant repositories remains to be investigated.

In general, during the design and implementation process, it is difficult to determine which features should be implemented, so that the software is reusable. It is a balancing act to find the right scope between a tailored solution and a software, that may be applied to answer further research questions. The advantages of Jupyter Notebooks become particularly evident in the exploratory process. Ideas may be quickly implemented and tested. But the execution sequence has to be taken into account to not overwrite variables. This also occurs when running a Notebook from another Notebook.

Initially, the framework should be implemented merely with Jupyter Notebooks. Six Notebooks should be implemented, three harvester Notebooks, one Notebook for the data cleansing and preparation, one for the analysis, and one to reproduce the study. Except the Notebook for the study execution, all Notebooks become quickly too complex and cluttered. This does not promote a good code quality. Therefore, the program logic was implemented as Python modules. Additionally, the features of an IDE are missing, like code linting and refactoring. As workaround to combine the interactivity of Jupyter Notebooks with the benefits of an IDE, it is suggested to develop Python modules in an IDE, and import them into the Notebook for execution.¹

Regarding the reproducibility, harvesting publications is hampered. Often an authentication token or registration is required. In the worst case, like for the ACM digital library no API is provided. This necessitates the use of a web crawler. They are not an appropriate

¹https://godatadriven.com/blog/write-less-terrible-code-with-jupyter-notebook/
7. Conclusions and Future Work

method for automated reuse. For instance, when the structure of a website is changed, the crawling method has to be modified. When using Scrapy with Jupyter Notebooks, a CrawlerProcess can be run only once. Then the Notebook kernel has to be restarted.

Another peculiarity in the publications harvesting process relates to arXiv. When searching for GitHub links, only GitHub references for the corresponding paper are returned that are mentioned in the summary, the summary details, or the arxiv comment. Further referenced repositories in the paper are lost. So only one category of research software is recorded, the others are lost. When harvesting ACM publications, the full text is scanned, but not the reference list.

Some of the repositories belong to two analysis groups. Especially, the membership in the ACM and arXiv group attracted attention. The arXiv publication may be the preprint of the ACM publication. The handling of preprints is unattended in the current framework. However, this double counting should not have a strong effect on the results. The condition of having a related research publication is met. It may be that the research software repository is regarded interdisciplinary, as no DOI metadata may be gathered for the preprint and the arXiv taxonomy is applied. In the current study this affects about 600 repositories, therefore, it must be decided whether it is worth to invest valuable time.

Repositories with a lifespan of more than 12 years attracted attention, as GitHub was introduced in 2008. So far it is unexplained how the first commit dates before 2008 are created. Maybe the repository was created on another hosting service and then migrated to GitHub. For instance, BitBucket stop the support for mercurial and introduced git. In this context, some developers switch to GitHub. This may also explain the negative lifespans, if errors occur during the migration process.

7.2 Future Work

The harvested data still hold a large amount of available information that may be further evaluated. So far, the lifespan of the repositories is analyzes on the basis of the three coarse categories GitHub, ACM, and arXiv. As the research software repositories have an assigned research subject, the lifespan may be considered regarding computer science and computational science individually. A closer look may be taken on the features of the repositories that are referenced in both ACM and arXiv publications, and itself contains a DOI. The information about the primary repository language are available and may be related to the research subjects.

Due to the increasing amount of data, the architecture of the current framework has to be reviewed regarding the aspects mentioned in Section 7.1. A conceivable implementation could be the use of Jupyter Notebooks only for visualization processes and hiding the program logic in separate modules. This design could also increase the performance and stability of the framework.

As already mentioned by Hasselbring et al. [2020b], the search space for research software repositories may be enlarged and the identification process refined. The data
basis for research software repositories may be extended in a number of ways. For one thing, further repository hosting services, like Bitbucket, GitLab, GNU Savanna, and Launchpad, may be included into the harvesting process. Due to the provided interface of the framework, this should not be a great deal of work. Another way is the addition of other digital libraries. Here, also the provided interface can be used.

Besides the direct harvesting of publications, in research software repositories the publications are not only referenced by a DOI. Links to the publication itself or BibTeX entries are used to refer to a related publication. Regarding the various appearances of references, it may laborious to cover them all individually. Moreover, new ideas from community driven initiatives for publishing research software, like papers with code, or collecting points for relevant research software, like RedditSOTA, steadily emerges.\textsuperscript{23} This may result in a unmanageable plethora of locations where to find research software. But instead of extending the search space to as many locations of research software as possible, the concept of the Research Software Observatory may be implemented as single point of contact for information, reuse, and metadata harvesting.

\footnote{\url{https://paperswithcode.com/}}
\footnote{\url{https://github.com/RedditSota/state-of-the-art-result-for-machine-learning-problems}}
Bibliography


Bibliography


Bibliography


Bibliography


Appendices
# Parameter for the study of the research software publication and sustainability

# set your authentication token if exists and required
authentication:
github: ""

# set the database and database table names
database:
name: fons
repo_collection: repositories
pub_collection: publications
rs_collection: researchSoftware

# set the name of the repository hosting service
# default value: github
repo_sources:
- github

# set the name of the digital library
# default value: arxiv
pub_sources:
- arxiv

# set the keyword(s)
# special characters that will be ignored in the search string:
# . , : ; / ' " " = ! ? # $ & + ^ | ~ < > ( ) { } [ ]
# default keywords for repositories: doi, doi+in:readme
# default keywords for publications: github.com
repo_keywords:
- doi
- doi+in:readme
pub_keywords:
- github.com
A. Configuration File

```plaintext
# set the dates for the repository search period
# default start and end date for reproduction: 2020-08-01,
    2020-10-12
start_date: "2020-08-01"
end_date: "2020-10-12"

# set the repository search interval in the format years|months|days = int
# default value: days=7
delta: days=7

# specify the information to be collected by the repositoryHarvester
repoHarvester:
  - search_repos
  - readme

# specify the steps to be executed by the research software identifier
rsidentifier:
  - new_publications
  - new_repositories
  - dois
  - repo_name
  - user_name
  - metadata
  - content
  - cleansing
  - commits
  - crossref
  - subject
  - subject_arxiv
```
Requirements

Listing B.1. Requirements file

```
1 backcall==0.2.0
2 certifi==2020.6.20
3 chardet==3.0.4
4 decorator==4.4.2
5 feedparser==6.0.1
6 idna==2.10
7 ipykernel==5.3.4
8 ipython==7.16.1
9 ipython-genutils==0.2.0
10 jedi==0.17.2
11 jupyter-client==6.1.7
12 jupyter-core==4.6.3
13 parso==0.7.1
14 pexpect==4.8.0
15 pickleshare==0.7.5
16 pkg-resources==0.0.0
17 prompt-toolkit==3.0.7
18 ptyprocess==0.6.0
19 Pygments==2.7.1
20 pymongo==3.11.0
21 python-dateutil==2.8.1
22 PyYAML==5.3.1
23 pyzmq==19.0.2
24 requests==2.24.0
25 sgmllib3k==1.0.0
26 six==1.15.0
27 tornado==6.0.4
28 traitlets==4.3.3
29 urllib3==1.25.10
30 wcwidth==0.2.5
```
Analysis

C.1 Research Subjects Overview

C.2 Research Fields
### C. Analysis

**Table C.1.** Research field overview with assigned research subject. (A) labels the research fields from the arXiv taxonomy

<table>
<thead>
<tr>
<th>Research Field</th>
<th>Research Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multidisciplinary</td>
<td>Multidisciplinary</td>
</tr>
<tr>
<td>Agricultural and Biological Sciences</td>
<td>Life Sciences</td>
</tr>
<tr>
<td>Arts and Humanities</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Biochemistry, Genetics and Molecular Biology</td>
<td>Life Sciences</td>
</tr>
<tr>
<td>Business, Management and Accounting</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Chemical Engineering</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Chemistry</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Computer Science (A,S)</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Decision Sciences</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Dentistry</td>
<td>Health Sciences</td>
</tr>
<tr>
<td>Earth and Planetary Sciences</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Economics (A)</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Economics, Econometrics and Finance</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Electrical Engineering and Systems Science (A)</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Energy</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Engineering</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Environmental Science</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Health Professions</td>
<td>Health Sciences</td>
</tr>
<tr>
<td>Immunology and Microbiology</td>
<td>Life Sciences</td>
</tr>
<tr>
<td>Materials Science</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Mathematics (A,S)</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Medicine</td>
<td>Health Sciences</td>
</tr>
<tr>
<td>Neuroscience</td>
<td>Life Sciences</td>
</tr>
<tr>
<td>Nursing</td>
<td>Health Sciences</td>
</tr>
<tr>
<td>Pharmacology, Toxicology and Pharmaceutics</td>
<td>Life Sciences</td>
</tr>
<tr>
<td>Physics (A)</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Physics and Astronomy</td>
<td>Physical Sciences</td>
</tr>
<tr>
<td>Psychology</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Quantitative Biology (A)</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Quantitative Finance (A)</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Social Sciences</td>
<td>Social Sciences</td>
</tr>
<tr>
<td>Statistics (A)</td>
<td>Health Sciences</td>
</tr>
<tr>
<td>Veterinary</td>
<td>Health Sciences</td>
</tr>
</tbody>
</table>
### Table C.2. Research field overview with the number of assigned repositories

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number of repositories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>42,922</td>
</tr>
<tr>
<td>Biochemistry, Genetics and Molecular Biology</td>
<td>10,019</td>
</tr>
<tr>
<td>Mathematics</td>
<td>7,277</td>
</tr>
<tr>
<td>Engineering</td>
<td>6,526</td>
</tr>
<tr>
<td>Agricultural and Biological Sciences</td>
<td>5,249</td>
</tr>
<tr>
<td>Earth and Planetary Sciences</td>
<td>4,326</td>
</tr>
<tr>
<td>Medicine</td>
<td>3,636</td>
</tr>
<tr>
<td>Environmental Science</td>
<td>2,813</td>
</tr>
<tr>
<td>Multidisciplinary</td>
<td>2,802</td>
</tr>
<tr>
<td>Chemistry</td>
<td>2,756</td>
</tr>
<tr>
<td>Decision Sciences</td>
<td>2,658</td>
</tr>
<tr>
<td>Physics and Astronomy</td>
<td>2,254</td>
</tr>
<tr>
<td>Social Sciences</td>
<td>2,120</td>
</tr>
<tr>
<td>Arts and Humanities</td>
<td>1,856</td>
</tr>
<tr>
<td>Neuroscience</td>
<td>1,756</td>
</tr>
<tr>
<td>Chemical Engineering</td>
<td>1,205</td>
</tr>
<tr>
<td>Immunology and Microbiology</td>
<td>1,178</td>
</tr>
<tr>
<td>Business, Management and Accounting</td>
<td>972</td>
</tr>
<tr>
<td>Psychology</td>
<td>762</td>
</tr>
<tr>
<td>Materials Science</td>
<td>715</td>
</tr>
<tr>
<td>Health Professions</td>
<td>677</td>
</tr>
<tr>
<td>Energy</td>
<td>519</td>
</tr>
<tr>
<td>Statistics</td>
<td>380</td>
</tr>
<tr>
<td>Economics, Econometrics and Finance</td>
<td>346</td>
</tr>
<tr>
<td>Astrophysics</td>
<td>270</td>
</tr>
<tr>
<td>Electrical Engineering and Systems Science</td>
<td>245</td>
</tr>
<tr>
<td>Pharmacology, Toxicology and Pharmaceutics</td>
<td>223</td>
</tr>
<tr>
<td>Quantitative Biology</td>
<td>124</td>
</tr>
<tr>
<td>Physics</td>
<td>72</td>
</tr>
<tr>
<td>Quantum Physics</td>
<td>68</td>
</tr>
<tr>
<td>High Energy Physics - Phenomenology</td>
<td>42</td>
</tr>
<tr>
<td>Nursing</td>
<td>37</td>
</tr>
<tr>
<td>Condensed Matter</td>
<td>31</td>
</tr>
<tr>
<td>Quantitative Finance</td>
<td>18</td>
</tr>
<tr>
<td>Veterinary</td>
<td>17</td>
</tr>
<tr>
<td>Economics</td>
<td>12</td>
</tr>
<tr>
<td>General Relativity and Quantum Cosmology</td>
<td>12</td>
</tr>
<tr>
<td>Dentistry</td>
<td>10</td>
</tr>
<tr>
<td>High Energy Physics - Theory</td>
<td>9</td>
</tr>
<tr>
<td>High Energy Physics - Lattice</td>
<td>8</td>
</tr>
<tr>
<td>Mathematical Physics</td>
<td>3</td>
</tr>
<tr>
<td>Nuclear Theory</td>
<td>3</td>
</tr>
<tr>
<td>High Energy Physics - Experiment</td>
<td>2</td>
</tr>
<tr>
<td>Nonlinear Sciences</td>
<td>2</td>
</tr>
</tbody>
</table>