Alexander Struck, Oliver Bertuch und Stephan Druskat

5.8 Forschungssoftware

1 Einführung und erste Einordnung

Die Erstellung und Verwendung von Forschungssoftware findet in der Regel innerhalb einer Forschungscommunity statt. Studien zeigen die große Bedeutung von Software für den wissenschaftlichen Erkenntnisprozess[1] und dass die an der Erstellung von Forschungssoftware beteiligten Personen im Unterschied zu kommerziellen und nicht-kommerziellen Softwareprodukten meist die Forschenden selbst sind.[2] Über die Dissemination der Software ist bekannt, dass sie meist über direkte Kontakte und wissenschaftliche Publikationen (die in der Regel auf ein Zielpublikum gerichtet sind) erfolgt,[3] was den Schluss zulässt, dass eine Nachnutzung über die Grenzen einer Disziplin hinaus nicht die Regel ist. Aufgrund der Finanzierungsstrukturen in der Wissenschaft ist eine Auftragsentwicklung durch die Software-Industrie eher selten.

Im Gegensatz zu Projekten in der Softwareindustrie unterliegt die Mehrheit der Forschungssoftware nicht dem finanziellen Zwang, Produktcharakter aufweisen zu müssen. Es handelt sich bei Forschungssoftware um ein Spektrum verschiedenster Projekte, das von codierter, purer, wissenschaftlicher Expertise bis zu Software reicht, die Forschungsprozesse unterstützt, dokumentiert und begleitet (z. B. Annotations- und Editionssoftware, elektronische Laborbücher, Repositoriums-Software, Peer Review-Systeme usw.). Möglich ist somit die Evolution einer Forschungssoftware von kleinen, datenzentrierten Codes über Prototypen bis zu Forschungssoftware-Infrastruktur.

Software, die wissenschaftliche Erkenntnisse generiert und die codierte wissenschaftliche Expertise enthält, ist auch selbst als valides Ergebnis der Forschung anzuerkennen, gleichwertig zu anderen Forschungsoutputs. Diese Erkenntnis ist über die letzten Jahre gereift und findet sich in Deutschland wieder in den Leitlinien zur Sicherung guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft (DFG),[4] sowie verschiedenen Statements und Leitlinien zu Open Science, z. B. das G6 Statement on Open Science.[5] Sie ist unter anderem begründet in der Komplexität der in

1 Schindler u. a. 2022.

2 Wiese, Polato u. Pinto 2020.

3 AlNoamany u. Borghi 2018.

4 https://wissenschaftliche-integritaet.de/kodex/ (25.10.2023).

5 https://os.helmholtz.de/assets/open_science/Downloads/G6_statement_on_Open_Science.pdf, https://os.helmholtz.de/open-science-in-helmholtz/open-science-policy/ (alle 25.10.2023).

Danksagung: Alexander Struck dankt für die Unterstützung des Exzellenzclusters „Matters of Activity. Image Space Material“, gefördert durch die Deutsche Forschungsgemeinschaft (DFG) im Rahmen der Exzellenzstrategie des Bundes und der Länder – EXC 2025 – 390648296.

Open Access. © 2024 bei den Autorinnen und Autoren, publiziert von De Gruyter. CC BY Dieses Werk ist lizenziert unter der Creative Commons Namensnennung 4.0 International Lizenz.
https://doi.org/10.1515/9783111046341-027

Forschungssoftware codierten theoretischen Konstrukte (u. a. Heuristiken, Methoden, Modelle).[6] Somit sind auch Bibliotheken wichtiger Bestandteil eines Ökosystems für Forschungssoftware.[7]

Aus dieser Entwicklung leiten sich potenzielle Aufgaben für Bibliotheken und Archive ab. Analog zur steigenden Bedeutung und Wahrnehmung von Forschungsdaten und deren Management sind Publikation, Archivierung, Nachweis, Recherche und Zitation von Forschungssoftware eine neue Herausforderung. Die folgenden Abschnitte analysieren und diskutieren diese im Detail. Über diese klassisch bibliothekarischen Aufgaben hinaus ist – in Abhängigkeit lokaler Gegebenheiten – die Ansiedlung von Expertise in Research Software Engineering eine zu diskutierende Fragestellung.

In keinem Fall können Bibliotheken Forschungssoftware wie eine Untermenge von Forschungsdaten behandeln: Daten sind Fakten und Beobachtungen, die Belege liefern.[8] Software ist das Resultat eines kreativen Prozesses, der ein Werkzeug für eine Aufgabe – z. B. mit Daten – hervorbringt. Software ist ausführbar, Daten sind es nicht. Oftmals ist Software ein Baustein für andere Software, insbesondere dann, wenn es der Organisation von komplexen Arbeitsabläufen mit Software und Daten dient. In der Regel wird Software nicht von Grund auf neu geschrieben, sondern verwendet Funktionalität verschiedener anderer Softwarepakete nach. Die Software und diese teils komplexen Abhängigkeiten unterliegen – im Gegensatz zu Daten – regelmäßigen Änderungen ihres Verhaltens und ihrer Schnittstellen.

2 Metadaten

Ähnlich wie bei Forschungsdaten existieren auch für Software Metadaten. Diese werden entweder aktiv aufgezeichnet oder entstehen als Nebenprodukte des Softwareentwicklungsprozesses. Grundsätzlich lassen sich verschiedene Metadatentypen für Software bestimmen, die sich als generische Metadaten und softwarespezifische Metadaten klassifizieren lassen.

Generische Metadatentypen beinhalten, auf Software bezogen, Metadaten, die auch für andere Forschungsobjekte existieren. Dazu gehören beispielsweise die Benennung der Software, Autorenschaft, rechtliche – insbesondere lizenzrechtliche – Angaben, Informationen zu Förderung, wissenschaftlichen Disziplinen und anderen Kategorien, Zitation, Veröffentlichung, Zugang, Beziehungen zu anderen Forschungsobjekten sowie Beschreibungen der Software.

Darüber hinaus lassen sich **softwarespezifische Metadaten** nur für Software definieren. Beispiele hierfür sind Informationen zu in Software nachgenutzten Soft-

6 Jay u. a. 2020.
7 Anzt u. a. 2020.
8 Lamprecht u. a. 2020.

warepaketen, verwendete Programmiersprachen, Versionsinformationen aus Versionskontrollsystemen und Versionsidentifikatoren, Anforderungsbeschreibungen für Laufzeitumgebungen, Hardware und Infrastruktur sowie softwarespezifische Code-, Qualitäts-, Entwicklungs- und Nutzungsmetriken.

Softwaremetadaten können in verschiedenen Modi und Formaten vorliegen. Für bibliothekarische Anwendungsfälle liegen Metadaten idealerweise persistent als Dateien in bekannten Datenserialisierungssprachen wie JSON, XML oder YAML vor und können so relativ einfach in Bibliothekssysteme übernommen werden, beispielsweise durch Konvertierung. Darüber hinaus werden Metadaten von Softwareprojekten nicht selten eingebettet in unstrukturierte Textdateien, die der Softwaredokumentation dienen, so zum Beispiel README-Dateien. Andere Metadatentypen sind transient und erst aus Entwicklungssystemen zu extrahieren, z. B. über programmatischen Zugriff auf Versionskontrollsysteme oder Web-APIs von Plattformen zur statischen Codeanalyse.

Einige Metadatenformate haben sich zu Communitystandards entwickelt. Dazu gehören beispielsweise schema.org[9] (in JSON-LD) zur Beschreibung von Webressourcen, CodeMeta[10] (in JSON-LD) zur generischen Beschreibung von Softwareartefakten und das Citation File Format[11] (in YAML) zur Bereitstellung von Zitationsmetadaten für Software. Andere Formate sind spezifisch für Programmiersprachen und Buildsysteme und werden für die Beschreibung von legalen Metadaten, Reife der Software, Schnittstellen, Softwareabhängigkeiten und Buildprozessen verwendet. Für Forschungssoftware kommen darüber hinaus infrastrukturseitig das DataCite Metadata-Schema[12] beziehungsweise dessen softwarespezifische Untermenge zum Einsatz. Für komplexe Forschungsobjekte, die auch Software beinhalten, werden RO-Crate[13] und das generischere BagIt[14] verwendet.

Softwaremetadaten sind zentraler Baustein der Prinzipien von FAIR for Research Software (FAIR4RS[15] und siehe Abschnitt 6 „FAIR4RS“ in diesem Kapitel). Als solcher sind sie für Bibliotheken als Managerinnen von Forschungsobjekten von höchster Relevanz. Die Aufnahme und Bereitstellung reicher Metadaten spezifisch für Forschungssoftware, über bibliographische Angaben hinaus, ist daher Kernbestandteil moderner bibliothekarischer Arbeit. Dazu gehört auch die Verwendung von Metadatenmodellen, die Software als eigenen Typ kennen und softwarespezifische Metadaten abbilden können.

9 https://schema.org/ (25.10.2023).
10 https://codemeta.github.io/ (25.10.2023).
11 Druskat u. a. 2021.
12 https://schema.datacite.org/ (25.10.2023).
13 https://www.researchobject.org/ro-crate/ (25.10.2023).
14 https://datatracker.ietf.org/doc/html/rfc8493 (25.10.2023).
15 Chue Hong u. a. 2022.

Forschungssoftwarekonzepte und -beschreibungen können potenziell in Ontologien formalisiert werden. Relevant sind hier die Software Description Ontology,[16] die Software Ontology,[17] SEON[18] und DOAP,[19] wobei diese teilweise domänenspezifisch sind. Für alle gilt, dass sie derzeit allgemein nicht viel Anwendung im Research Software Engineering erfahren. EDAM[20] hingegen findet standardmäßig Anwendung in bioinformatischen Datenanalyse-Workflows.

3 Publikationen

3.1 Einführung

Bibliotheken und Archive kuratieren ihre Bestände, teils liegen Sammelaufträge zugrunde. Monographien, Zeitschriften und andere textbasierte Publikationen, die diese Institutionen in ihren Beständen pflegen, weisen einen vergleichsweise langen Lebenszyklus auf. Methoden wie Lektorat und Peer Review – in der Regel durch die Verlage organisiert – garantieren eine gewisse Qualität (die durch die Regeln der Verlage definiert ist). Die Literaturbeschaffung orientiert sich an etablierten Qualitätsmerkmalen. Diese klassischen Methoden der Bestandspflege in Bibliotheken sind für Forschungssoftware nur bedingt anwendbar.

Erstens unterliegt Forschungssoftware anderen Lebenszyklen. Da die Entwicklung dieser Software nicht nur Ergebnis, sondern immanenter Teil des dynamischen Forschungsprozesses ist, entstehen während der Laufzeit eines Forschungsprojekts viele Versionen in teils schneller Folge. Ziel von Open Science ist nicht nur die Offenheit der Forschung, sondern auch eine Steigerung der Reproduzierbarkeit von Ergebnissen. Dazu ist unerlässlich, verwendete Versionen einer Software mit Hilfe von persistenten Identifikatoren zu referenzieren.

Zweitens entstehen diese Versionen während des laufenden Forschungsprozesses, weshalb Lektorat oder Peer Review keine passenden Qualitätssicherungsprozesse sind. Einerseits sind diese zu langsam: Es wäre schlicht inakzeptabel, auf die Publikation einer Version mehrere Monate warten zu müssen. Andererseits existiert derzeit keine Infrastruktur mit der notwendigen fachlichen Expertise sowohl in Forschungsdisziplin als auch in Softwareentwicklung.[21]

16 https://w3id.org/okn/o/sd (25.10.2023).
17 https://github.com/allysonlister/swo (25.10.2023).
18 http://se-on.org/ (25.10.2023).
19 https://github.com/ewilderj/doap/wiki (25.10.2023).
20 https://edamontology.org (25.10.2023).
21 Alternativen und ihre Limitationen finden sich im Abschnitt 5 „Qualität & Kuration“ in diesem Kapitel.

Um die Nachnutzbarkeit und Wiederauffindbarkeit von Forschungssoftware zu unterstützen, könnten Bibliotheken den Aufbau von Software-Verzeichnissen mit ihrer Expertise in Kuration unterstützen. Beispiele für solche Verzeichnisse sind die domänenspezifische Astrophysics Source Code Library,[22] das internationale Research Software Directory[23] und das Helmholtz Software Directory.[24]

In Anbetracht der Vielzahl von Forschungssoftware, die heute existiert, aber unzureichend aufbereitet und verfügbar gemacht und damit im Sinne von Open Science unsichtbar ist, können Bibliotheken den initialen Bestandsaufbau der Verzeichnisse ebenfalls unterstützen, indem sie bei der Publikation der Software und ihrer Katalogisierung behilflich sind. Hierzu zählen Beratungsangebote und Hinweise auf gute Praxis,[25] Leitlinien und Leitfäden genauso wie konkrete Hilfestellung und ggf. Infrastruktur.

3.2 Praxis

Für den deutschen Wissenschafts- und Bibliotheksbetrieb leitet sich aus den Leitlinien der DFG[26] die starke Empfehlung zur Publikation von Forschungssoftware ab. Leitlinie 13: „Selbst programmierte Software wird unter Angabe des Quellcodes öffentlich zugänglich gemacht." Leitlinie 7: „Der Quellcode von öffentlich zugänglicher Software muss persistent, zitierbar und dokumentiert sein." Gegebenenfalls ergeben sich aus den Bedingungen von Förderern, Verlagen oder Institutionen weitere bindende Verpflichtungen. Grundsätzlich ist festzustellen, dass sich die derzeitige Praxis der Verfügbarmachung von Software von einem bibliothekarischen Verständnis der Forschungspublikation unterscheidet.

In der Vergangenheit wurde Software lediglich in Textpublikationen beschrieben. Dies sichert Autor:innen akademische Anerkennung, ist aber für den wissenschaftlichen Nachweis der Software selbst klar ungeeignet. Softwarereferenzen nahmen wenn überhaupt die Form von Links zu Webseiten an. Teilweise wurde Quellcode als Supplement einer Textpublikation hinzugefügt. Auch hier findet kein Nachweis der Software als eigenständiges Forschungsergebnis statt. Als Brückentechnologie haben sich sogenannte Software-Journals entwickelt, z. B. Journal of Open Source Software, Journal of Open Research Software, IPOL Journal.[27] Hier wird Software selbst mit kurzen Begleittexten eingereicht und üblicherweise technisch begutachtet. Angenommene Software wird in geeigneten Publikationsrepositorien verfügbar gemacht. Dort

22 http://www.ascl.net/ (25.10.2023).
23 https://research-software-directory.org/ (25.10.2023).
24 https://helmholtz.software/ (25.10.2023).
25 S. Katz u. a. 2020.
26 https://wissenschaftliche-integritaet.de/kodex/ (25.10.2023).
27 Für weitere Beispiele siehe die Liste des britischen Software Sustainability Institute: https://www.software.ac.uk/which-journals-should-i-publish-my-software (25.10.2023).

erstellte PIDs werden zusammen mit dem Verweis auf Source Code und Begleittext als „Software Paper“ veröffentlicht. So wird wissenschaftliche Anerkennung durch Verwendung traditioneller Formen sichergestellt. Aber auch diese Form der Publikation wird den dynamischen Entwicklungsprozessen und der Frequenz der Versionierung von Software nicht gerecht.

Derzeit bildet sich die Publikation von Software auf allgemein zugänglichen Publikationsplattformen wie beispielsweise InvenioRDM (Zenodo), Dataverse und anderen als bevorzugte Methode heraus. Diese Plattformen vergeben PIDs für einzelne Softwareversionen und teilweise für Versionssammlungen. Sie stellen außerdem maschinenlesbare Metadaten und Landing Pages für Versionen bereit. Zudem bieten sie verschieden mächtige Bestandssuchen an. Diese Plattformen werden teilweise auch als disziplinäre oder institutionelle Repositorien betrieben. Sie fördern die akademische Anerkennung von Softwareautor:innen, erfordern aber entsprechende Eignung, um Software in ihren Eigenheiten repräsentieren und ihre Auffindbarkeit sicherstellen zu können. Formales fachliches (Software) Peer Review ist bislang nicht institutionalisiert, sondern findet informell vor oder nach Publikation über Werkzeuge aus dem Open Peer Review oder der Softwareentwicklung (z. B. Issues) statt.

Eine gängige Praxis ist derzeit noch die Publikation durch Distribution, die sich nicht mit einem bibliothekarischen Verständnis von Publikation deckt. Studien zeigen, dass in zahlreichen Artikeln Referenzen auf Plattformen zur Zusammenarbeit in der Software-Entwicklung zu finden sind.[28] Ein simples Teilen der eigenen Software mit der wissenschaftlichen Community auf diesem Weg ist jedoch keine wissenschaftliche Publikation im engeren Sinne, sondern entspricht einer Distribution (was oftmals ignorierte juristische Verpflichtungen mit sich bringt). Es ist weder die langfristige Verfügbarkeit noch die Autorenschaft sicher geklärt und nachvollziehbar.

In der bibliothekarischen Praxis äußern Forschende Bedenken, die sie von einer Publikation ihrer Forschungssoftware abhalten, wie sie auch in verschiedenen Studien (z. B. AlNoamany und Borghi[29]) beobachtet wurden. Diese sind analog zu den Gründen, aus denen Daten nicht publiziert werden: Furcht vor Wettbewerbsnachteilen, Qualitätsmängeln oder für Laien nicht einschätzbaren juristischen Problemen. Gegebenenfalls kann hier gezieltes Zurückhalten bestimmter Softwareartefakte, bis zur reinen Metadaten-Publikation, dem Problem begegnen.

Ferner sind sich Forschende der juristischen Konsequenzen ihres Handelns bei der Weitergabe von Software oftmals nicht bewusst. Sofern Bibliotheken in die Publikation einer Software, die gleichzeitig eine öffentlich zugängliche Distribution beinhaltet, involviert sind, sollten mindestens Hinweise auf die Beachtung von Urheber- und Verwertungsrecht, Patentrecht, wirtschaftlicher Verwertung, Außenwirtschaftsrecht und Lizenzkompatibilität gegeben werden; idealerweise sollte auf der Einhaltung von Frei-

28 Z. B. Schindler u. a. 2022; Hasselbring u. a. 2020.

29 AlNoamany u. Borghi 2018.

gabeprozessen und Leitlinien der wissenschaftlichen Institution zu Software bestanden werden.

Oftmals stellen sich Forschende die Frage, wie sie mit gemischten Inhalten umgehen sollen, da in der Regel nicht nur Software, sondern auch Daten an der Generierung von Ergebnissen beteiligt sind. Sofern die Daten und/oder die Software eigenständig nachgenutzt werden könnten (egal durch wen oder wie wahrscheinlich), sollten getrennte Publikationen für beides erstellt werden. Spielt die Software nur eine untergeordnete Rolle oder ist nur im Kontext des Datensatzes verwendbar, kann sie ggf. in der Datenpublikation subsumiert werden. Wächst die Software weiter, kann sie jederzeit in eine eigenständige Softwarepublikation überführt werden. Spielen die in einer Softwarepublikation enthaltenen Daten nur eine untergeordnete Rolle (z. B. als Testdaten, etc.), kann mit diesen analog zum umgekehrten, obigen Fall verfahren werden.

Die Erstellung von Softwarepublikationen erfordert von den Forschenden Zusatzarbeit in Bezug auf Metadaten und Lizenzierung. Einfach strukturierte Freigabeprozesse zur Publikation als Open Source sind an wissenschaftlichen Institutionen noch keine Selbstverständlichkeit. Die Sammlung der notwendigen Metadaten ist oftmals noch Handarbeit. Projekte wie SURESOFT[30] und HERMES,[31] an denen Bibliotheken beteiligt sind, begegnen dem mit Automatisierung. Ebenso existiert mit der Integration von Github und Zenodo[32] für eine in der Forschung häufig verwendete Code-Hosting-Plattform eine Möglichkeit der automatischen Publikation (und Archivierung).[33]

3.3 Infrastruktur

In den letzten Jahren haben viele wissenschaftliche Bibliotheken eigenständig oder in enger Zusammenarbeit mit Rechenzentren Publikationsplattformen und/oder Nachweisverzeichnisse für Forschungsdaten aufgebaut. Die analoge Verwendung dieser vorhandenen Infrastruktur für Publikation und/oder Nachweis von Forschungssoftware liegt nahe für Anwendungsfälle, in denen keine disziplinspezifischen Lösungen existieren.

Neben der Einhaltung von generellen Empfehlungen für Software-Publikationsrepositorien[34] sind ggf. weitere Anpassungen in der Infrastruktur notwendig, damit die Publikationsplattformen bereit für Forschungssoftware sind. Sofern die Plattformen die nachfolgende Eigenschaften nicht oder nur teilweise erfüllen, schränkt dies die Auffindbarkeit von Software ein:

30 https://www.tu-braunschweig.de/suresoft (25.10.2023).

31 Druskat u. a. 2022.

32 https://docs.github.com/de/repositories/archiving-a-github-repository/referencing-and-citing-content#issuing-a-persistent-identifier-for-your-repository-with-zenodo (25.10.2023).

33 Eine Sammlung weiterer Ansätze findet sich in Druskat u. a. 2022.

34 S. Garijo u. a. 2022.

1. Unterscheidung zwischen Daten- und Software-Artefakten (und ggf. weiteren)
   a) Innerhalb der Publikationsplattform muss für Nutzende sowohl beim Einliefern als auch beim Abruf der Fokus einer gegebenen Menge von Metadaten und ggf. zugehöriger Dateien ersichtlich bzw. definierbar sein.
   b) Ohne eine solche klare Klassifizierung fehlt in der technischen Umsetzung eine eindeutige Möglichkeit der Fallunterscheidung in Präsentation und Verarbeitung.
   c) Das Repositorium sollte bei Hinzufügen von Quellcode-Dateien dies zum einen bemerken und zum anderen im Falle einer (geplanten) Datenpublikation nachfragen, ob die Subsumierung beabsichtigt ist und ggf. mehr Optionen anbieten.
2. Unterstützung für Software-Metadaten
   a) Neben der Aufnahme, Speicherung, Indizierung, Durchsuchung und Darstellung (z. B. auf Basis des CodeMeta Schemas) müssen diese als CodeMeta exportiert und exponiert werden können.
   b) Ebenso ist eine Übersetzung in Schemata von Anbietern persistenter Identifikatoren unter Wahrung der Unterscheidung (siehe Punkt 1) notwendig (Beispiel: DataCite-Schema unterstützt die Registrierung als Software).
   c) Wünschenswert ist die Übersetzung und Bereitstellung der Software-Metadaten als XML-RDF via OAI-PMH und als JSON-LD für SEO-Zwecke. Dies fördert die Auffindbarkeit, da mit diesen gängigen Techniken Verzeichnisse und Suchmaschinen direkt befüllt und aktualisiert werden können.
3. Unterstützung von persistenten Identifikatoren für „Konzept“ und „Version“
   a) Aus FAIR4RS[35] ergibt sich die Anforderung, eine Softwarepublikation sowohl insgesamt (als „Konzept“) als auch ihre einzelnen „Versionen“ durch persistente Identifikatoren eindeutig referenzierbar zu machen.
   b) Die Versionsnummern des Datensatzes auf einer Publikationsplattform und der Software an sich dürfen sich unterscheiden. Persistente Identifikatoren sollen keine semantische Bedeutung aufweisen und die Version der Software ist in den Metadaten hinterlegt. Dies ist auch dem Umstand dienlich, dass das Versionierungsschema einer Software sich von der Publikationsplattform erheblich unterscheiden kann.
4. Unterstützung von Software-Lizenzen
   a) Lizenzen für Daten sind nur im Ausnahmefall für Software anwendbar. Zum einen wird mit der Erstellung einer Software im deutschen Rechtsrahmen regelmäßig eine ausreichende Schöpfungshöhe für einen urheberrechtlichen Schutz erreicht (oft im Gegensatz zu Daten als reinen Fakten). Zum anderen ergeben sich für Software andere Schutzbedürfnisse, z. B. in Bezug auf Patente, Garantien, Haftung usw.

35 Chue Hong u. a. 2022.

b) Eine Publikationsplattform muss daher für Software-Einreichung eine geeignete (und ggf. kuratierte) Auswahl von Software-Lizenzen akzeptieren und entsprechend annotieren. Eine Liste findet sich z. B. via SPDX.[36] Die Unterstützung von Creative Commons ist nicht ausreichend.

Es gibt gute Gründe für Institutionen, eigene interne Plattformen zur kollaborativen Entwicklung von Software zu betreiben. Auf Basis von zum Beispiel GitLab usw. lassen sich solche Plattformen erstellen, wenn z. B. bestimmte Software zu kritisch ist, um sie öffentlich zugänglich zu machen. Auf dieser technischen Grundlage können Bibliotheken Forschende bei Publikation und Archivierung unterstützen.

Die Automatisierung der Publikation von Software mit Werkzeugen wie HERMES[37] kann für die Institution, der die Bibliothek angehört, weiter vereinfacht werden, wenn durch die Bibliothek Vorlagen zur Nachnutzung bereitgestellt werden. Ebenso ist es möglich, Kuration von Metadaten und die Einlieferung in die eigene Publikationsplattform zu erleichtern, indem passende Erweiterungsmodule durch die Bibliothek entwickelt und bereitgestellt werden.

3.4 Archivierung

Für die Langzeitarchivierung von Software und deren Kuration finden sich zwei unterschiedliche Ansätze. Diese unterscheiden sich in ihrem technischen und personellen Aufwand stark.

Der erste Ansatz beschränkt sich auf die Archivierung des Quellcodes von Software. Als Beispiel sei hier das Software Heritage Archive[38] genannt. Eine Auswahl der zu archivierenden Quellcodes findet explizit nicht statt, Ziel ist die Archivierung des weltweiten Softwarebestandes. Dort existieren auch Kopien von andernorts gelöschtem Quellcode. Eine Kuration findet auf technischer Ebene durch Deduplizierung gleichen Quellcodes statt. Die reine Verfügbarkeit von Quellcode allein ist jedoch kein Garant für eine Ausführbarkeit der Software innerhalb der üblichen Zeiträume von Langzeitarchiven.

Der zweite Ansatz bemüht sich um die Ausführbarkeit der archivierten Software. Dies ist ungleich aufwändiger, da die technischen Nebenbedingungen eingehalten werden müssen. Insbesondere wenn kein Quellcode mit archiviert werden konnte, müssen für eine erneute Ausführung gleiche Hardware, Betriebssystem und Softwareabhängigkeiten vorhanden sein oder emuliert werden können. Das von der Yale University Library gestartete und vom Software Preservation Network[39] unter-

36 https://spdx.dev/ (25.10.2023).
37 https://docs.software-metadata.pub/ (25.10.2023).
38 https://www.softwareheritage.org/ (25.10.2023).
39 https://www.softwarepreservationnetwork.org/ (25.10.2023).

stützte Projekt EaaSI[40] ist hier ein Beispiel. Durch die in den USA vorhandene Urheberrechtsklausel des „Fair Use“ ist diese Möglichkeit nicht ohne weiteres in Europa nachstellbar.

3.5 Virtualisierungen & ausführbare Umgebungen

Seit ca. 15 Jahren existiert die Möglichkeit zur Virtualisierung von Ausführungsumgebungen und der sogenannten Containerisierung aller Abhängigkeiten. Apptainer- bzw. Singularity- und Docker-Container machen diese Technologie leicht zugänglich und verbreiten sich zunehmend auch in Forschung und Entwicklung. Insbesondere Apptainer (ehem. Singularity) spielt für die Reproduzierbarkeit von Forschungsergebnissen in der Wissenschaft eine wichtige Rolle. Für Container Images entwickeln sich derzeit sogenannte Registries, die der Nachnutzung dienen und eine Publikationsform darstellen können.

Seit knapp 10 Jahren werden außerdem web-basierte Ausführungsumgebungen, wie Jupyter Notebooks, vorangebracht, die – in einer Kombination von Text und Source Code – die Nachvollziehbarkeit und Evaluation von software-basierten Forschungsergebnissen vereinfachen. Deren Archivierung ist durch umfassenden Einsatz von Open Source Technologie erleichtert.

Es gibt weiterhin Forschungssoftware, die direkte Abhängigkeiten zu notwendiger Hardware aufweist; als Beispiel sei High-Performance-Computing (HPC) genannt. Trotz Fortschritten in der Virtualisierung oder Abstrahierung kann Emulation spezieller Hardware weiterhin eine Rolle für die Ausführbarkeit von Software spielen.

3.6 Wiederauffindbarkeit

Die Suche nach Forschungssoftware ist eine Herausforderung für Forschende. Abgesehen von populären Web-Suchmaschinen ist oft nicht bekannt, wo und wie Information Retrieval nach Software erfolgversprechend ist. Das kollegiale bzw. soziale Netzwerk spielt eine gewisse Rolle, wie auch disziplinenspezifische Software Reviews. (Research) Software Engineers suchen noch eher in Git-basierten Code-Kollaborationsplattformen.[41]

Für einzelne Disziplinen ist die in Forschungsliteratur erwähnte Software generell zu einem Drittel und spezifische Versionen zu 90 % nicht mehr verfügbar.[42] Das ist in Teilen der gewachsenen Praxis der Verfügbarmachung geschuldet.

40 https://www.eaasi.info/ (25.10.2023).
41 Vgl. Hucka u. Graham 2018.
42 Howison u. Bullard 2016.

Gleichzeitig weist re3data.org bereits mehrere Hundert Plattformen nach, die Software-Applikationen und/oder Source Code enthalten. Sofern diese Plattformen etablierte APIs anbieten, können Metasuchdienste wie BASE-search.net derartige Angebote bündeln und durchsuchbar machen. Allerdings werden bei Aggregatoren oft nur die meist spärlich vorhandenen Metadaten indexiert. Existierende Metadatenschemata kommen bei gemischten Inhalten, z. B. in institutionellen Repositorien, an ihre Grenzen. Das wirkt sich auf die Wiederauffindbarkeit aus. Erschließungswerkzeuge für Softwarepublikationen wie kontrollierte Vokabulare, Taxonomien oder Ontologien werden in einigen Disziplinen erarbeitet.

Existierende Software wird aus verschiedenen Gründen auch nicht recherchiert. Teils aus der Annahme heraus, es existiere keine bestehende Lösung für ein Problem. Die Erstellung eines State-of-the-art-Überblicks ist aufwändig und durch die heterogene Publikationspraxis teils unvollständig. Gefundene Software müsste evaluiert werden, was in Forschungsanträgen selten berücksichtigt wird.

Verschiedene Ansätze sollen diese Situation verbessern. Publikationen mit umfangreichen Metadaten nach den FAIR4RS-Prinzipien können die Wiederauffindbarkeit erhöhen. Sauberes Zitieren von Software(-Publikationen) in der Literatur kann deren Nutzung verdeutlichen und erhöhen. Softwarekataloge, wie das Research Software Directory (RSD)[43] weisen in Disziplinen, für Organisationen oder (inter-)national andernorts publizierte Forschungssoftware nach und erhöhen so deren Sichtbarkeit. Da auch web-basierte Forschungsplattformen in Teilen softwarebasiert sind, indexieren einige Registries wie EOSC neben Tools auch Services. Andere Plattformen wie swMATH[44] weisen Literatur und z. B. Zitationen rund um Software nach und machen diese sichtbarer.

Im Abschnitt zu Publikationen erwähnte Plattformen bereichern die Diversität möglicher Standorte relevanter Software. So sind ausgewählte Softwarepakete auch in Betriebssystemdistributionen, sogenannten Paketmanagern und programmiersprachenspezifischen Repositorien verfügbar.

4 Zitierung von Software

Management und Nachweis von Forschungsartefakten, so auch Forschungssoftware, sind Kernauftrag wissenschaftlicher Bibliotheken. Forschungssoftware als eigenständiges, zitierfähiges Ergebnis des Forschungsprozesses bringt neue Herausforderungen mit sich. Die Unterschiede zwischen Software und anderen Forschungsartefakten (vor allem Textpublikationen und Forschungsdaten) in Entstehung, Management und Publikation schlagen sich auch in der Zitierung nieder.

43 https://research-software-directory.org/ (25.10.2023).

44 https://swmath.org/ (25.10.2023).

Hierbei muss zunächst die Anerkennung von Software als zitierfähiges Artefakt sichergestellt werden:[45] Die Software selbst muss zitierbar sein und zitiert werden. Behelfsmäßige Zitierung traditioneller Textpublikationen, die die Software selbst oder ihre Verwendung lediglich für eine spezifische Momentaufnahme eines im Allgemeinen dynamischen Softwareentwicklungsprozesses beschreiben, steht guter wissenschaftlicher Praxis eindeutig entgegen. Über übliche bibliographische Metadaten zu Autorschaft, Name bzw. Titel des Artefakts und Publikationsdatum hinaus spielt insbesondere der Nachweis einzelner Softwareversionen eine wichtige Rolle. Dieser ist Voraussetzung für die Reproduzierbarkeit komputationell erlangter Forschungsergebnisse. Durch die besonderen Eigenheiten von Softwarepublikation gegenüber der Publikation von Texten muss Zugänglichkeit von Forschungssoftware über Zitation auch dann sichergestellt werden können, wenn keine persistenten Identifier (z. B. DOIs) für eine Softwareversion zur Verfügung stehen. Dies bedarf spezifizierter Metadaten, z. B. für die eindeutige Identifikation von Quellcode-Ressourcen.

Im Gegensatz zu traditionellen Textpublikationen[46] sind die Autorschaft von Software und ihre Voraussetzungen noch nicht formal definiert. So können sich bestimmte Rollen im komplexen Softwareentwicklungsprozess für die Autorschaft qualifizieren, ohne dass sie direkt Beiträge zu Quellcode, Dokumentation oder anderen Artefakten nachweisen können. Auf der anderen Seite fallen nicht alle Personen, die nachweislich Beiträge zu Quellcode oder Dokumentation liefern, zwangsläufig unter eine Definition von Autorschaft.

Durch häufig fehlende Kuration und wenig formalisierte Publikationsprozesse obliegt die Bereitstellung von korrekten und vollständigen Zitationsmetadaten für Forschungssoftware zunächst ausschließlich den Softwareautor:innen selbst. Diese erfolgt zunehmend über die Bereitstellung von Dateien im Citation File Format (CFF)[47] zusammen mit dem Source Code der jeweiligen Softwareversionen. An dieser Stelle können Bibliotheken die Verwendung dieser Metadaten für eigene Bestände, Datenbanken und Interfaces sicherstellen. Gleiches gilt für die Verknüpfung einzelner Versionen einer Software. Das Publikationsrepository Zenodo[48] löst dies beispielsweise über sogenannte Concept DOIs, die verschiedene Versionen des semantisch selben Artefakts zusammenfassen und zur jeweils letzten veröffentlichten Version auflösen.

Sind die relevanten Metadaten zur Zitierung von Forschungssoftware Bestandteil eines Bibliothekskataloges geworden, wird dieser im Allgemeinen dem Publikum in Formaten zur Wiederverwendung in der eigenen Arbeit angeboten. Für Textpublikationen sind das üblicherweise: 1) vorformatierte Referenzen in bestimmten Zitationsstilen, 2) generische Zitationsmetadatenformate wie BibTeX und 3) Metadatenformate für Re-

45 Smith, Katz u. Niemeyer 2016.

46 S. https://www.icmje.org/recommendations/browse/roles-and-responsibilities/defining-the-role-of-authors-and-contributors.html (25.10.2023) und Guidelines von Zeitschriften.

47 Druskat u. a. 2021.

48 https://zenodo.org/ (25.10.2023).

ferenzmanagementsoftware. Einige dieser Werkzeuge haben bereits Anpassungen für die Abbildung von Softwarezitationsmetadaten vorgenommen und unterstützen den direkten Import von CFF. BibTeX hat zwar einen Alias *@software* für seinen generischen Typ *@misc*, dieser validiert aber nicht semantisch eindeutig die für Zitierung von Software notwendigen Metadaten (z. B. Versionsangaben). Das Paket biblatex-software[49] bietet hier für BibLaTeX eine Alternative, die die notwendigen Metadatenfelder umsetzt und auch Softwareteile zitierbar macht.

5 Qualität & Kuration

Wenn Bibliotheken Ansprechpartnerinnen für Publikation und eventuelle Archivierung, Bestandspflege, Nachweis und Auffindbarkeit von Forschungssoftware werden, stellt sich die Frage, wie Qualitätskriterien von Softwareartefakten und deren Metadaten zu beurteilen und Qualität gegebenenfalls zu sichern sind. Qualitätssicherung ist hier sowohl bei der Erstellung der entsprechenden Softwareartefakte im Research Software Engineering als auch im Kontext der FAIR Principles for Research Software zu betrachten.[50]

5.1 Softwarequalität und (Research) Software Engineering

Als Unterklasse von Software ist die Qualitätssicherung für Forschungssoftware grundsätzlich Aufgabe des angewandten Software Engineering. Dabei unterliegt Forschungssoftware zunächst den hier üblichen Prozessen und Methoden:[51] Requirements Engineering, Design, Entwicklungsprozess, Test, Wartung usw. Analog sollten die entsprechenden Methoden der Software-Qualitätssicherung Anwendung finden: funktionsorientiertes, kontroll- und datenflussorientiertes, dynamisches und gegebenenfalls modellbasiertes Testen, Messung und Bewertung durch statische Codeanalyse, Softwareinspektionen und Code-Reviews, symbolisches Testen und formale Korrektheitsbeweise usw. Die Software Engineering-Literatur[52] stellt jedoch allgemein für angewandtes Software Engineering eine Diskrepanz zwischen theoretischem Wissen über Software-Qualitätssicherung und der Praxis fest.

Im Bezug auf Forschungssoftware kommen hier mindestens zwei Faktoren erschwerend hinzu. Zum einen wird Forschungssoftware teilweise experimentell entwickelt und ist in diesen Fällen nicht auf das Erreichen hoher Reifegrade ausgelegt.

49 https://ctan.org/pkg/biblatex-software (25.10.2023).
50 Chue Hong u. a. 2022.
51 https://www.computer.org/education/bodies-of-knowledge/software-engineering (25.10.2023).
52 Z. B. Liggesmeyer 2009.

Darüber hinaus birgt ihre Domäne besondere Risiken in Bezug auf die Unbekanntheit der mit Forschungssoftware untersuchten Phänomene, inhärente Komplexität, sich konstant ändernde Wissensgrundlagen, hohe Datenabhängigkeit, sowie inhärent unbekannte Resultate und somit das Fehlen deterministischer Testfälle.[53]

Zum anderen stellt die Entwicklung von Forschungssoftware Anforderungen an Entwickler:innen, die häufig über das Fachwissen traditioneller Software Engineers hinausgehen, teilweise im Bereich numerischer Algorithmen und High Performance Computing, vor allem aber in der Anwendungsdomäne selbst, der entsprechenden Fachwissenschaft. Hier hat sich in den letzten Jahren eine neue, heterogene Rolle herausgebildet, die diese Lücke schließen kann: die des Research Software Engineers (RSE). RSEs sind oft selbst fachwissenschaftlich sowie entweder formell oder informell informatisch ausgebildet und bewegen sich zwischen Fachwissenschaft und Software Engineering. Dabei stehen programmierende Forschende am einen und Forschung unterstützende Software Engineers am anderen Ende des Spektrums. RSEs sind in besonderer Weise dazu geeignet, fachwissenschaftlich begründete Korrektheit von Software herzustellen, zu beurteilen und zu gewährleisten. Auch sie sind allerdings von im akademischen System nach wie vor üblichen befristeten Arbeitsverträgen betroffen. Dies hat zur Folge, dass bestimmten Merkmalen von Forschungssoftwarequalität besondere Wichtigkeit zukommt, beispielsweise der Wartbarkeit, Dokumentation und Wiederverwendbarkeit. In Disziplinen, die komplexe Berechnung, Simulation oder Modellierung durchführen, zählt dazu Performanz als wichtiges Qualitätsmerkmal.

Research Software Engineering gewinnt zunehmend an Bedeutung, was sich auch in der Gründung von Fachgesellschaften niederschlägt. Für Forschungssoftware heißt das, dass im besten Fall ausgebildete Software Engineers mit Domänenwissen sich auf die Qualitätssicherung der Software konzentrieren. In anderen Fällen kann es aber auch bedeuten, dass Forschende mit wenigen Kenntnissen – und eventuell wenig Interesse – in Software Engineering Softwareartefakte schaffen, auf deren Grundlage Forschungsergebnisse generiert und publiziert werden. Damit ist Softwarequalität in der Forschung über den Selbstzweck hinaus ein entscheidender Faktor bei der Sicherung von Wissen. Dies zeigt sich insbesondere in der Frage der Reproduzierbarkeit. Wo sich informatisches Software Engineering die Reproduzierbarkeit von Softwareartefakten aus Entwicklungsprozessen zum Ziel setzt, geht es beim Research Software Engineering zusätzlich um die Reproduzierbarkeit von Forschungsergebnissen und somit die Sicherung von allgemeinem Wissen.[54]

Im Software Engineering sind die Qualitätsanforderungen auch abhängig vom Reifegrad der Software. So werden Prototypen anders bemessen und bewertet als marktreife Produkte. In der Forschung, wo sowohl kurze Skripte zur Datenanalyse als auch hochkomplexe und ausgereifte Modelle, Simulationssoftware und Forschungssoftware-

53 Carver, Hong u. Thiruvathukal 2017.

54 Hasselbring u. a. 2020.

Frameworks zum Einsatz kommen, spielt Softwarequalität mindestens im Bezug auf die Korrektheit und Reproduzierbarkeit von Forschungsergebnissen auch unabhängig vom Reifegrad eine Rolle. Nichtsdestotrotz werden auch in der Forschung verschiedene Anwendungsszenarien in Qualitätsanforderungen abgebildet, wie beispielsweise die Klassifizierung von Applikationen in institutionellen Richtlinien für Software Engineering.[55]

Qualitätssicherung findet bei Forschungssoftware derzeit also nicht wie üblich durch klassisches Peer Review statt. Vielmehr wird Software zunächst verifiziert und validiert: verifiziert beispielsweise im Hinblick auf Fehlerfreiheit und die formale Umsetzung von Anforderungen, validiert im Hinblick auf die Erfüllung von Nutzungserwartungen. Verifiziert wird im (Research) Software Engineering üblicherweise durch Testen und statische Codeanalyse, wobei beides automatisiert und kontinuierlich erfolgen kann. Validiert wird über Code Reviews und Inspektionen, wobei hier im Sinne des Peer Reviews auch die Erfüllung fachlicher Anforderungen erfasst werden kann. Die fachliche Qualitätssicherung findet somit meist in den Softwareprojekten selbst statt. Ausnahme hierbei sind Qualitätsaudits, die aber selten und nur bei hoher Kritikalität – wie beispielsweise in der Luft- und Raumfahrtforschung oder der Medizin – stattfinden. Qualitätssicherung von Forschungssoftwaremetadaten kann jedoch auch von Bibliotheken übernommen werden, beispielsweise im Kontext von Softwaremanagementplänen, aber auch in der Kuration der Softwaremetadaten selbst.

5.2 Standards

Während industrielle Softwareentwicklung teilweise an relevante Standards zu Qualitätsanforderungen und -management wie ISO/IEC 90003 und ISO/IEC 25000 gebunden ist, finden diese Normen im Research Software Engineering noch wenig Anwendung. In einigen Bereichen haben sich Communitystandards wie CFF gebildet, die bereits in Softwarekatalogen und Publikationsplattformen benutzt werden.

Grundsätzlich deckt der von der IEEE Computer Society herausgegebene *Guide to the Software Engineering Body of Knowledge*[56] auch Forschungssoftware ab, ist aber auf Grund der Ausbildungssituation vieler RSEs nicht bekannt und findet daher insbesondere in experimentellen Softwarevorhaben selten Anwendung.

Signifikanten Einfluss auf die Forschungssoftwarelandschaft haben Entwicklungen aus der Free/Libre Open Source Community, beispielsweise im Hinblick auf die Verwendung von kollaborativen Versionskontrollplattformen (GitHub, GitLab u. a.), Lizenzierung, die Verwendung spezifischer Dokumentation (README und LICENSE-

55 https://rse.dlr.de/01_guidelines.html (25.10.2023).

56 https://www.computer.org/education/bodies-of-knowledge/software-engineering (25.10.2023).

Dateien) und kollaborative Entwicklungsworkflows, etwa durch Integration von Codebeiträgen Dritter.

6 FAIR4RS

Die FAIR Principles for Research Data sind auf Software kaum anwendbar und wurden daher von der FAIR4RS-Arbeitsgruppe der RDA für Forschungssoftware als FAIR4RS Leitlinien[57] angepasst.

Sie stellen ein Assessment hinsichtlich bestimmter Qualitäten dar. Aus den Prinzipien ist ersichtlich, dass FAIRe Software nicht zwingend Open Source-Lizenzen unterliegt bzw. kostenfrei verfügbar sein muss. Damit wird dem Wunsch einiger Forschungseinrichtungen nach kommerzieller Verwertbarkeit Rechnung getragen.

- *Findable* bezieht sich auf Menschen und Maschinen. Das erfordert Persistent Identifiers (PIDs), Versionierung und die Publikation mit sinnvollen Metadaten in akzeptierten Schemata und Vokabularien. Sowohl die Software als auch ihre Metadaten sollen FAIR sein, damit z. B. Indexing Services ermöglicht werden.
- *Accessible* ist das Prinzip der Zugänglichkeit, wobei die Software per PID und offenen, freien Kommunikationsprotokollen beziehbar sein muss und die Metadaten verfügbar bleiben, auch wenn das nicht mehr auf die Software zutrifft.
- *Interoperable* ist Software, sobald sie mit anderer Software (Meta-)Daten z. B. über definierte/standardisierte APIs oder Formate austauscht. Beim Austausch kommen domänenspezifische Definitionen zum Tragen, die die Bedürfnisse der jeweiligen Community erfüllen. FAIRe Software referenziert andere (z. B. nachgenutzte) Research Objects.
- Das Prinzip *Reusable* verlangt, dass Software (im Kontext der jeweiligen Disziplin) verständlich, veränderbar und (nach-)nutzbar ist. Das setzt eindeutige Lizenzen und detaillierte Provenienzinformationen voraus.

Die Prüfung auf FAIRness erfolgt bei Forschungsdaten und Software noch inkonsistent und mittels verschiedener (teils automatisierter) Werkzeuge.[58]

57 Chue Hong u. a. 2022.

58 Z. B. https://gitlab.bsc.es/inb/elixir/software-observatory/FAIRsoft_ETL (25.10.2023); Spaaks u. a. 2022.

7 Open Science & Policies

Forschungssoftware gerät auch deshalb zunehmend in den Fokus von Bibliotheken, weil sie erstens historisch durch die Free-/Libre-Open-Source-Bewegung inhärent starke Anknüpfungspunkte zu Open Science hat und zweitens die bibliothekarische Beschäftigung mit Software durch Policies explizit gefordert wird.

Open Science fordert, dass Daten, Methoden und Ergebnisse von Forschung der allgemeinen Öffentlichkeit auch international zur Wiederverwendung, Weiterverbreitung und Vervielfältigung zugänglich gemacht werden. Für Forschungssoftware bedeutet dies vor allem die öffentliche Entwicklung von Software und ihre Lizenzierung unter Open Source-Lizenzen[59]. Argumentativ wird dies insbesondere von durch die öffentliche Hand (mit-)geförderte Entwicklung von Forschungssoftware gefordert, wie von der Initiative Public Money Public Code.[60] Sofern nicht beispielsweise institutionelle Richtlinien zur Verwertung von Software dagegensprechen, fördert offene Lizenzierung Nachnutzung und Reproduzierbarkeit von Forschung. Des Weiteren müssen für die Beschaffung dieser Software keine Mittel der Forschungsorganisation aufgewendet werden. Aber auch die Ermöglichung von Wiederverwendung durch Dokumentation und Bereitstellung von Programmierschnittstellen und Runtimes kann in weiterem Sinne unter Open Science mit Bezug auf Forschungssoftware gefasst werden. Um in diesem Sinne bestmögliche Zugänglichkeit zu gewährleisten, muss Forschungssoftware publiziert werden.

Vorgaben machen hier Policies aus der Politik, von Forschungsförderern und Institutionen. Auf europäischer Ebene ist dies zum Beispiel die European Research Area Policy Agenda, über die Open Science auch für Forschungssoftware etwa in der European Open Science Cloud implementiert wird. Die Deutsche Forschungsgemeinschaft fordert in Leitlinie 13 zur Sicherung guter wissenschaftlicher Praxis,[61] dass selbst programmierte Software unter Angabe des Quellcodes öffentlich zugänglich gemacht wird. Auch die Open Science Policy der Helmholtz-Gemeinschaft Deutscher Forschungszentren[62] schreibt vor, dass Software, die zur Bewertung von Forschungsergebnissen notwendig ist, offen publiziert wird. Darüber hinaus sieht sie wissenschaftliche Evaluation auf der Grundlage FAIR publizierter Forschungssoftware vor. Bibliotheken sind in diesem Kontext wichtige Akteurinnen, die sich nicht nur beispielsweise durch Bereitstellung von Publikationsinfrastruktur, Schulungen, Software- und Metadatenkuration, Bereitstellung von Katalogen und verwandten Aktivitäten die Erfüllung dieser Anforderungen ermöglichen können, sondern sich auch in die Entwicklung von Leitlinien einbringen sollten.

59 https://opensource.org/ (25.10.2023).

60 https://publiccode.eu/ (25.10.2023).

61 https://wissenschaftliche-integritaet.de/kodex/herstellung-von-offentlichem-zugang-zu-forschungsergebnissen/ (25.10.2023).

62 https://os.helmholtz.de/open-science-in-helmholtz/open-science-policy/ (25.10.2023).

8 Akteure & Ressourcen

Im anglo-amerikanischen Raum werden sogenannte Software Sustainability Institutes (SSI)[63] etabliert, die (inter-)national Einfluss auf die Wahrnehmung, das Management, die Qualität von Software und die Entwickelnden nehmen. Parallel bilden sich zumeist nationale Communities von Research Software Engineers (RSE),[64] die international zusammenarbeiten. Weitere Beispiele für international agierende Communities sind die o. g. Arbeitsgruppe FAIR4RS mit ihren Veröffentlichungen[65] oder die FORCE11 Working Groups.[66]

Die Research Software Alliance (ReSA)[67] engagiert sich (als Pendant zur RDA) für die noch notwendige Wertschätzung und Weiterentwicklung des Forschungssoftware-Ökosystems. Neben *Library* und *Data* existiert auch eine *Software* Carpentry Community, die Forschende weiterbildet.[68]

Als Finanziererin von Grundlagenforschung fördert die DFG die Erstellung von Software in Forschungsprojekten, jedoch nur selten deren Weiterentwicklung oder Wartung.[69] Die Verstetigung von Softwareprojekten wird bei Forschungsorganisationen oder der Gemeinschaft der Nutzenden gesehen. International ist NumFOCUS[70] ein Sponsor für die Weiterentwicklung ausgewählter wichtiger Forschungssoftware.

Wissenschaftliche Bibliotheken engagieren sich (teils in Kooperation mit Rechenzentren) beim Betrieb von oft multidisziplinären institutionellen Repositorien und der Kuration des Bestandes. Neben Texten, Medien und Daten werden dort auch Applikationssoftware oder Quellcode hinterlegt. Gleichzeitig existieren domänen-spezifische Bibliotheken wie die Astrophysics Source Code Library (ASCL),[71] die sich um den Nachweis von Forschungssoftware verdient machen.

Einzelne Bibliotheken bieten bereits Dienstleistungen bei der Erstellung oder dem Management von Software an. Beispielhaft sei hier die SUB Göttingen mit ihrer Gruppe „Software und Service-Entwicklung“ genannt. Dort werden Portale, Visualisierungen, Datenmodelle, digitale Editionen und mehr entwickelt. Diese (personelle) Infrastruktur deckt einen stetig steigenden Bedarf an Unterstützung in der Forschung. Ähnlich organisiert sind an (inter-)nationalen Forschungseinrichtungen aufgebaute RSE-Gruppen – teils in Kooperation mit der Bibliothek – die Beratung, Softwareentwicklung, Training und andere Dienste anbieten. Damit verbessern sich u. a. Software-

63 Z. B. https://software.ac.uk/ (25.10.2023).
64 Z. B. de-RSE – Gesellschaft für Forschungssoftware: https://de-rse.org/ (25.10.2023).
65 https://zenodo.org/communities/fair4rs (25.10.2023).
66 https://force11.org/groups (25.10.2023).
67 https://www.researchsoft.org/ (25.10.2023).
68 https://software-carpentry.org/ (25.10.2023).
69 https://www.dfg.de/en/research_funding/announcements_proposals/2022/info_wissenschaft_22_85/ (25.10.2023).
70 https://numfocus.org/sponsored-projects (25.10.2023).
71 http://ascl.net/ (25.10.2023).

qualität, Nachhaltigkeit, Zufriedenheit der Mitarbeitenden, Projektanträge, aber gleichzeitig erhöht sich auch der Verwaltungsaufwand.[72] Als ein weiteres Beispiel sei die Max Planck Digital Library (MPDL) genannt, die ein Template für einen Software Management Plan (SMP) entwickelt und im RDMOrganizer[73] bereitgestellt hat.

9 Zukünftige Entwicklungen

Absehbar scheint, dass wissenschaftliche Verlage vermehrt neben der Verfügbarkeit von Daten auch die von Software einfordern werden, um zumindest im Reviewprozess die Möglichkeit der Prüfung zu ermöglichen. Ob sich kompetente Reviewer finden, wird die Zukunft zeigen. Hier kann das Prinzip "Open Review" von Texten auf Software ausgedehnt werden. Initiativen wie CodeCheck[74] stellen einen wichtigen Beitrag für die Verbesserung und Reproduzierbarkeit von Forschungssoftware dar.

Forschungsförderer sehen Datenmanagementpläne (DMP) in manchen Bereichen als erforderlich an. Vereinzelt werden auch schon Softwaremanagementpläne (SMP) angefragt, was zunehmen wird, wenn man die Entwicklung entsprechender SMP in einzelnen Forschungsorganisationen in Betracht zieht. Bibliotheken engagieren sich bereits bei DMP. Da SMP ähnliche Vorteile für die Qualität und das Management von Software bieten, erwarten wir trotz des zusätzlichen administrativen Aufwands eine weitere Verbreitung.

Bibliotheken werden in Zukunft mehr Forschungssoftware nachweisen bzw. zugänglich machen. Derzeit existiert eine offene Diskussion zu möglichen Qualitätsindikatoren bei Forschungssoftware, die neben Software Engineering-Standards und FAIR4RS Prinzipien auch Faktoren wie Impact einbeziehen können. Hier könnte die Expertise wissenschaftlicher Bibliotheken einen relevanten Beitrag leisten.

Literatur

AlNoamany, Yasmin u. John A. Borghi: Towards computational reproducibility: researcher perspectives on the use and sharing of software. In: PeerJ. Computer science 4 (2018), e163. https://doi.org/10.7717/peerj-cs.163.

Anzt, Hartwig, Felix Bach, Stephan Druskat, Frank Löffler, Axel Loewe, Bernhard Y. Renard, Gunnar Seemann, Alexander Struck, Elke Achhammer, Piush Aggarwal, Franziska Appel, Michael Bader, Lutz Brusch, Christian Busse, Chourdakis, Gerasimos, Piotr Wojciech Dabrowski, Peter Ebert, Bernd Flemisch, Sven Friedl, Bernadette Fritzsch, Maximilian D. Funk, Volker Gast, Florian Goth, Jean-Noël Grad, Jan Hegewald, Sibylle Hermann, Florian Hohmann, Stephan Janosch, Dominik Kutra, Jan

72 Schima-Voigt 2023.

73 https://github.com/rdmorganiser/rdmo-catalog (25.10.2023).

74 https://codecheck.org.uk/ (25.10.2023).

Linxweiler, Thilo Muth, Wolfgang Peters-Kottig, Fabian Rack, Fabian H. C. Raters; Stephan Rave, Guido Reina, Malte Reißig, Timo Ropinski, Joerg Schaarschmidt, Heidi Seibold, Jan P. Thiele, Benjamin Uekermann, Stefan Unger u. Rudolf Weeber: An environment for sustainable research software in Germany and beyond: current state, open challenges, and call for action. In: F1000Research 9 (2020), 295. https://doi.org/10.12688/f1000research.23224.2.

Carver, Jeffrey C., Neil P. Chue Hong u. George K. Thiruvathukal (Hrsg.): Software engineering for science. Boca Raton, London, New York: CRC Press Taylor & Francis Group (Chapman & Hall/CRC) 2017.

Chue Hong, Neil P., Daniel S. Katz, Michelle Barker, Anna-Lena Lamprecht, Carlos Martinez, Fotis E. Psomopoulos, Jen Harrow, Leyla Jael Castro, Morane Gruenpeter, Paula Andrea Martinez u. Tom Honeyman: FAIR Principles for Research Software (FAIR4RS Principles). 2022. https://doi.org/10.15497/RDA00065.

Druskat, Stephan, Oliver Bertuch, Guido Juckeland, Oliver Knodel u. Tobias Schlauch: Software publications with rich metadata: state of the art, automated workflows and HERMES concept. 2022. https://doi.org/10.48550/arXiv.2201.09015.

Druskat, Stephan, Jurriaan H.Spaaks, Neil Chue Hong, Robert Haines, James Baker, Spencer Bliven, Egon Willighagen, David Pérez-Suárez u. Alexander Konovalov: Citation File Format. In: Zenodo (2021). https://doi.org/10.5281/zenodo.1003149.

Garijo, Daniel, Hervé Ménager, Lorraine Hwang, Ana Trisovic, Michael Hucka, Thomas Morrell u. Alice Allen: Nine best practices for research software registries and repositories. In: PeerJ. Computer science 8 (2022), e1023. https://doi.org/10.7717/peerj-cs.1023.

Hasselbring, Wilhelm, Leslie Carr, Simon Hettrick, Heather Packer u. Thanassis Tiropanis: Open Source Research Software. In: Computer 53 (2020), H. 8, S. 84–88. https://doi.org/10.1109/MC.2020.2998235.

Howison, James u. Julia Bullard: Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. In: Journal of the Association for Information Science and Technology 67 (2016), H. 9, S. 2137–2155. https://doi.org/10.1002/asi.23538.

Hucka, M. u. M. J. Graham: Software search is not a science, even among scientists: A survey of how scientists and engineers find software. In: Journal of Systems and Software 141 (2018), S. 171–191. https://doi.org/10.1016/j.jss.2018.03.047.

Jay, Caroline, Robert Haines, Daniel S. Katz, Jeffrey C. Carver, Sandra Gesing, Steven R. Brandt, James Howison, Anshu Dubey, James C. Phillips, Hui Wan u. Matthew J. Turk: The challenges of theory-software translation. In: F1000Research 9 (2020), 1192. https://doi.org/10.12688/f1000research.25561.1.

Katz, Daniel S., Neil P. Chue Hong, Tim Clark, August Muench, Shelley Stall, Daina Bouquin, Matthew Cannon, Scott Edmunds, Telli Faez, Patricia Feeney, Martin Fenner, Michael Friedman, Gerry Grenier, Melissa Harrison, Joerg Heber, Adam Leary, Catriona MacCallum, Hollydawn Murray, Erika Pastrana, Katherine Perry, Douglas Schuster, Martina Stockhause u. Jake Yeston: Recognizing the value of software: a software citation guide. In: F1000Research 9 (2020), 1257. https://doi.org/10.12688/f1000research.26932.2.

Lamprecht, Anna-Lena, Leyla Garcia, Mateusz Kuzak, Carlos Martinez, Ricardo Arcila, Eva Del Martin Pico, Victoria Del Dominguez Angel, Stephanie van de Sandt, Jon Ison, Paula Andrea Martinez, Peter McQuilton, Alfonso Valencia, Jennifer Harrow, Fotis Psomopoulos, Josep Ll.Gelpi, Neil Chue Hong, Carole Goble u. Salvador Capella-Gutierrez: Towards FAIR principles for research software. In: Data Science 3 (2020), H. 1, S. 37–59. https://doi.org/10.3233/DS-190026.

Liggesmeyer, Peter: Software-Qualität: Testen, Analysieren und Verifizieren von Software. 2. Aufl. Heidelberg: Spektrum Akademischer Verlag 2009.

Schima-Voigt, Kristine: Research Software Engineering als Service an einer wissenschaftlichen Bibliothek. In: Zenodo (2023). https://doi.org/10.5281/zenodo.7727987.

Schindler, David, Felix Bensmann, Stefan Dietze u. Frank Krüger: The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central. In: PeerJ. Computer science 8 (2022), e835. https://doi.org/10.7717/peerj-cs.835.

Smith, Arfon M., Daniel S. Katz, Kyle E. Niemeyer u.FORCE11 Software Citation Working Group: Software citation principles. In: PeerJ Computer Science 2 (2016), e86. https://doi.org/10.7717/peerj-cs.86.

Spaaks, Jurriaan H., Stefan Verhoeven, Erik Tjong Kim Sang, Faruk Diblen, Carlos Martinez-Ortiz, Edidiong Etuk, Mateusz Kuzak, Ben Werkhoven, Abel Soares Siqueira, Shyam Saladi u. Andrew Holding: howfairis. In: Zenodo (2022). https://doi.org/10.5281/zenodo.4017908.

Wiese, Igor, Ivanilton Polato u. Gustavo Pinto: Naming the Pain in Developing Scientific Software. In: IEEE Software 37 (2020), H. 4, S 75–82. https://doi.org/10.1109/MS.2019.2899838.