Gütekriterien und Handlungsempfehlungen für die Entwicklung von Forschungssoftware in der Kommunikations- und Medienwissenschaft

Mario Haim*

Die Entwicklung von Forschungssoftware ist für die empirische Kommunikations- und Medienwissenschaft (KMW) aufgrund sozialer, gesetzlicher, normativer und technologischer Veränderungen unabdingbar. Forschungssoftware umfasst dabei Skripte und Programme, die für den Zweck der Forschung und innerhalb des Fachs entwickelt werden, deren Entwicklung in der KMW jedoch, im Gegensatz zu einigen anderen Disziplinen, häufig innerhalb einzelner Forschungsprojekte und nicht selten durch (autodidaktisch) programmierende Forschende erfolgt – ein Umstand, der zwar Innovation fördert, gleichzeitig aber einer Institutionalisierung von Forschungssoftware entgegensteht. Dieser Beitrag leitet daher aus in dieser Hinsicht fortschrittlicheren Disziplinen neun Gütekriterien für Forschungssoftware in der KMW ab: Forschungssoftware sollte demnach zugänglich, anschlussfähig, verständlich, nachvollziehbar, autonom, strukturiert, verifiziert, umsichtig und nutzbar sein. Darauf aufbauend schlägt der Beitrag vier Handlungsempfehlungen vor, um diese Güte auch nachhaltig zu institutionalisieren: frühzeitige und gestärkte Methodenausbildung, angemessene Möglichkeiten der Sichtbarkeit und Anerkennung, mehr und passendere Förderformate sowie institutionalisierte Anreize, die eine professionelle Auseinandersetzung mit Forschungssoftware auch in Karriereoptionen übersetzen.

Schlüsselwörter: Methoden, Qualität, Nachhaltigkeit, Computational Communication Science, Computational Social Science

Quality Criteria and Recommendations for Developing Research Software in Communication Science

The development of research software in communication science has become inevitable. Constant changes of social, legal, normative, and technological predispositions render today's media and communication a moving research target, requiring perfectly suited methodological approaches and respective research software. Research software are thereby understood as scripts and applications developed within the field and as means for research. In communication science, research software has previously been developed within individual projects and oftentimes by individual (self-taught) developing researchers. While such individuality is certainly beneficial to innovation, it also contradicts the sustainability and thus institutionalization of research software. Building on findings and experiences from other disciplines, this article lists nine quality criteria to foster a more sustainable research-software landscape within communication science. That is, research software should be accessible, connectable, comprehensible, traceable, autonomous, structured, verified, cautious, and usable. In addition, this article compiles four recommendations to sustainably institutionalize high-

* Jun.-Prof. Dr. Mario Haim, Universität Leipzig, Institut für Kommunikations- und Medienwissenschaft, Nikolaistr. 27-29, Leipzig, Deutschland, mario.haim@uni-leipzig.de.
Der Autor bedankt sich bei den drei anonymen Gutachtenden sowie den Mitgliedern der DGPuK-Arbeitsgruppe Forschungssoftware für Anregungen und Feedback, insbesondere bei Alessandro Belli, Annett Heft, Andreas Hepp, Florian Hohmann, Jakob Jünger, Erik Koenen und Julian Unkel.

quality research software. That is, early and strengthened methodological training throughout curricula, various opportunities for visibility and recognition of research software, more and more appropriate funding formats to develop and maintain high-quality research software, and institutionalized incentives that help translate a professional approach to research software into academic career options.

Keywords: methods, quality, sustainability, computational communication science, computational social science

Die Erforschung digitaler Medien- und Informationsumgebungen unterliegt einem anhaltenden Wandel. Mindestens vier Strömungen sind dabei von zentraler Relevanz: (1) Soziale Veränderungen der Mediennutzung rücken immer neue Datenarten in den Fokus (z. B. personalisierte Newsfeeds, Popularitätshinweise, Geo-Informationen; vgl. auch Hepp, 2016); (2) gesetzliche Veränderungen sehen bisweilen Anpassungen der Datenhandhabung vor (z. B. DSGVO, Scraping; vgl. auch Rat für Sozial- und Wirtschaftsdaten, 2019); (3) normative Veränderungen verlangen immer häufiger nach nachhaltigerem Datenmanagement (z. B. Anforderungen bei Fördermitteln, Open-Science-Bestrebungen; vgl. auch Nosek et al., 2015); (4) technologische Veränderungen sorgen regelmäßig für immer neue und effizientere Möglichkeiten, etwa für die Datenerhebung (z. B. agentenbasierte Methoden, Tracking; vgl. auch Haim, 2020), erfordern gleichsam aber eine ständige Anpassung, etwa durch veränderte Datenverfügbarkeiten (z. B. API-Beschränkungen, Zugangsbarrieren; vgl. auch Freelon, 2018).

Diese Entwicklungen machen eine anhaltende Adaption von Forschungsdesigns und dem Umgang mit Daten nötig. Die Kommunikations- und Medienwissenschaft (KMW) ist entsprechend laufend gefordert, ihre Forschungssoftware anzupassen. Dieser Bedarf führt in jüngerer Zeit zu einer stetig wachsenden (van Atteveldt et al., 2019) und bisweilen fragmentierten (für die Journalismusforschung vgl. z. B. Haim & Zamith, 2019) Landschaft an Forschungssoftware, über die im Rahmen der zunehmend populären „Computational Communication Science" als Teil einer „Computational Social Science" international diskutiert wird (Lazer et al., 2020; van Atteveldt et al., 2019). Forschungssoftware wird je nach Anforderungen und verfügbaren Ressourcen vielerorts projektbezogen entwickelt, dabei aber nur gelegentlich veröffentlicht und anderen zur Verfügung gestellt, kaum durch Dritte begutachtet, und die Wartung oftmals mit Projektabschluss, spätestens aber mit dem Ausscheiden zentral Beteiligter, beendet. Gegen diese projektbezogene Einzelfallentwicklung ist zunächst wenig einzuwenden. Im Gegenteil: Innovative Vorstöße und explorative Bemühungen sind für den Fortschritt essenziell (van Atteveldt & Peng, 2018). Gleichzeitig bedarf es perspektivisch einer professionelleren Forschungssoftwareentwicklung im Fach (Haim, 2018), um sowohl die Güte *neuer* als auch den methodischen Fortschritt *bestehender* Forschungssoftware sicherzustellen.

Vor diesem Hintergrund wird Forschungssoftware hier sehr breit verstanden, als innerhalb der Fachgemeinde entwickelte Computerprogramme für jegliche Zwecke entlang des Forschungsprozesses. Forschungssoftware kann also für die Allgemeinheit entwickelte R-Pakete (z. B. Chan et al., 2020; Unkel, 2019) und (Web-)Applikationen (z. B. Hepp et al., 2018; Jünger & Keyling, 2017; Leiner, 2014) umfassen, ebenso wie spezifische Skripte der Datenauswertung einzelner Projekte, solange sie nur innerhalb des Fachs und für das Fach entwickelt wurden. Diese Einschränkung „innerhalb des Fachs und für das Fach" ist dabei nötig, um Handlungsempfehlungen an verschiedene Zielgruppen innerhalb der KMW geben zu können, die weder die Kompetenz der eigenen Disziplin noch die der Entwicklungsintention überschreiten. So folgen Gütekriterien

außerhalb der KMW bisweilen anderen Prämissen, etwa wenn es um die Lizenzierung genutzter Drittsoftware für die kommerzielle Weiterverwendung, die Effektivität entwickelter Algorithmen für die informatische Medizin oder die Effizienz der entwickelten Algorithmen für den informatischen Maschinenbau geht.

Dieser Beitrag diskutiert Gütekriterien für Forschungssoftware und konkrete Handlungsempfehlungen für verschiedene Zielgruppen und Stakeholder innerhalb der KMW. Dafür werden zunächst gängige Praktiken der wissenschaftlichen Softwareentwicklung anderer Disziplinen betrachtet, um daraus vier Pfeiler mit insgesamt neun Gütekriterien für das Fach abzuleiten. Anschließend werden diese Gütekriterien in konkrete Handlungsempfehlungen übertragen, wobei insbesondere auf die gängige Methodenausbildung, die Planung, Förderung, Entwicklung und Veröffentlichung von Forschungssoftware sowie die damit einhergehende Anerkennung eingegangen wird.

1. Gütekriterien

Forschungssoftware ist qua definitionem insbesondere in der Informatik Alltag. Dort diskutierte Gütekriterien sind primär auf die *Validierbarkeit* möglichst belastbar und effizient entwickelten Quellcodes sowie auf die *Nachhaltigkeit* entwickelter Forschungssoftware ausgelegt. Daneben existieren in Informatik-nahen interdisziplinären Fächern lebhafte Diskussionen um die Güte der jeweils fachspezifischen Forschungssoftware, etwa in der Bioinformatik („Computational Biology"), dem informatischen Maschinenbau („Computational Engineering Science") oder der informatischen Medizin („Computational Medicine") (Crouch et al., 2013). Von der *Validierbarkeit* abgesehen sind in diesen Disziplinen insbesondere Aspekte der *Replizierbarkeit* von großem Interesse. Darüber hinaus und quer zu diesen fachdisziplinären Diskursen finden sich immer häufiger fachübergreifende Diskussionen rund um die Open-Science-Bewegung (z. B. Dienlin et al., 2020; Lewis, 2020). Zentral diskutierte Kriterien sind dabei erneut die *Replizierbarkeit* sowie insbesondere die *Transparenz*. Ferner sehen auch die im Auftrag der Europäischen Kommission erarbeiteten „FAIR"-Prinzipien (findable, accessible, interoperable, reusable) akademischer Erzeugnisse insbesondere *Transparenz*, *Replizierbarkeit* und *Nachhaltigkeit* vor (Almeida et al., 2017; Hasselbring et al., 2020; Mons et al., 2017).

Diese insgesamt vier Diskussionsstränge – Transparenz, Replizierbarkeit, Validierbarkeit und Nachhaltigkeit – werden hier als Pfeiler professioneller Forschungssoftware verstanden. Innerhalb dieser Pfeiler sind für unterschiedliche Disziplinen unterschiedliche Kriterien relevant. So wird etwa unter Validierbarkeit im informatischen Maschinenbau die Sachrichtigkeit einzelner Software-Bestandteile priorisiert (Storer, 2017), während die Bioinformatik unter Validierbarkeit zunächst die Entwicklung durchdachter und nachvollziehbarer Testszenarien priorisiert (Zook et al., 2017). Entsprechend werden im Folgenden neun Gütekriterien für Forschungssoftware deklariert, die sich primär an den Anforderungen und Gegebenheiten der KMW orientieren (vgl. auch Tabelle 1).

1.1 Kriterien der Transparenz

Forschungssoftware in der KMW ist transparent, wenn sie zugänglich, anschlussfähig und verständlich ist.

Zugänglich meint zunächst die auffindbare und persistente Veröffentlichung in etablierten Repositorien (z. B. Harvard Dataverse, OSF; vgl. Stodden & Miguez, 2014). Zur Veröffentlichung gehören dabei neben dem Code auch für den Einsatz der Forschungs-

software nötige Dokumentationen, Daten, Materialien, Analysen und Beispiele (Bowman & Keene, 2018; Lewis, 2020). Widersprechen etwa rechtliche Rahmenbedingungen (z. B. des geistigen Eigentums oder des Schutzes der Privatsphäre) einer Offenlegung, soll die Zugänglichkeit der Forschungssoftware darunter nicht leiden; für solche Fälle sei an dieser Stelle auf entsprechende Werkzeuge zur Anonymisierung verwiesen (Dienlin et al., 2020; van Atteveldt et al., 2020). Zugänglich meint ferner die Auffindbarkeit veröffentlichter Forschungssoftware, um darüber in Austausch treten zu können. Dies lässt sich etwa mithilfe regelmäßiger Überblicke verfügbarer Forschungssoftware unterstützen (Storer, 2017), etwa in Form von Review-Beiträgen oder „Kürzlich erschienen"-Überblicken in den gängigen Fachzeitschriften. So würde über bewährte Kanäle auch jener Teil der Forschungsgemeinschaft auf neue Forschungssoftware aufmerksam, der bislang kaum Berührungspunkte mit den etablierten Repositorien hat.

Transparente Forschungssoftware soll zudem *anschlussfähig* sein, sich also an Normen und Standards orientieren. Standards ergeben sich dabei aus Anforderungen der Stakeholder (z. B. Drittmittelgebende, Fachzeitschriften; vgl. Blohowiak et al., 2013; DFG, 2019), die etwa nach bestimmten Technologien verlangen. Als Normen im Sinne gemeinhin akzeptierter Regeln und Leitlinien für die Forschungssoftware haben sich in der wissenschaftlichen Softwareentwicklung und nicht zuletzt in der KMW insbesondere die Nutzung von Python und R als Programmiersprachen etabliert, innerhalb von R finden etwa die tidyverse-Werkzeuge immer größere Verbreitung (Unkel, 2020; van Atteveldt et al., 2019; Wickham et al., 2019). Ferner haben sich Normen zur Formatierung von Code, also etwa der genutzten Einrückungen und Zeilenumbrüche (z. B. „PEP 8" in Python) (Nosek et al., 2015), herausgebildet, denen oftmals durch die Nutzung integrierter Entwicklungsumgebungen (IDE) oder entsprechender Pakete beigekommen werden kann (z. B. RStudio und formatR für R, PyCharm und PEP-8 für Python). Anschlussfähige Forschungssoftware stellt damit ein Mindestmaß an Lesbarkeit sicher und schafft gleichzeitig einheitlichere Anforderungen an das Vorwissen. Und obwohl sie damit eine leichte Einschränkung der Bewegungsfreiheit bedeutet, zumal im kreativen Umfeld der Softwareentwicklung, ist die Anschlussfähigkeit essenzieller Bestandteil professioneller Forschungssoftware, sowohl für die verständliche Nutzung bestehenden Codes (Nosek et al., 2015) als auch für die Ausbildung des akademischen Nachwuchses.

Zu einer *verständlichen* Forschungssoftware tragen außerdem die Gestaltung von Code und Daten sowie eine umfassende Dokumentation bei. So sollen Datenbezeichnungen eindeutig, mögliche Ausprägungen dokumentiert und Datensätze auf die Analyse zugeschnitten sein (Lewis, 2020; Wilson et al., 2017). Code ist im besten Fall selbsterklärend, was sich durch sinnvoll sprechende Bezeichnungen von Variablen und Funktionen auszeichnet (Storer, 2017), und gehört in jedem Fall dokumentiert (Lewis, 2020), idealerweise mit den in einer Programmiersprache üblichen Auszeichnungen (z. B. roxygen2 in R, docstrings in Python). Die Dokumentation umfasst dabei einleitende Kommentare zur Funktionalität eines Codes und beschreibt mindestens benötigte Parameter und erwartbare Rück- und Ausgabewerte (Wilson et al., 2017).

1.2 Kriterien der Replizierbarkeit

Forschungssoftware in der KMW ist replizierbar, wenn sie nachvollziehbar und autonom ist.

Nachvollziehbar meint die lückenlose Beschreibung für Außenstehende, also etwa die Schritte eines Auswertungsskripts von den Rohdaten bis zur finalen Analyse, die Generierung eines Modells von der Vorverarbeitung bis zur Validierung oder die Funktionsweise einer Anwendung von Anforderungen an die Laufzeitumgebung bis zur Be-

schreibung der Ausgabe. Damit einhergeht, sofern rechtlich möglich, die Veröffentlichung benötigter Ressourcen, etwa der Rohdaten selbst (Gentzkow & Shapiro, 2014). Eine lückenlose Beschreibung einer Forschungssoftware umfasst ferner Verweise auf alle eingesetzten Werkzeuge (z. B. Python- oder R-Pakete) sowie die Versionsnummern selbiger (Wilson et al., 2014) – eine Anforderung, die mittlerweile auch erste Fachzeitschriften an Veröffentlichungen stellen (z. B. Social Science Computer Review).

Idealtypisch ermöglicht eine Forschungssoftware außerdem eine *autonome* Replikation (Wilson et al., 2014, 2017). Im Fall eines Auswertungsskripts enthält die veröffentlichte Forschungssoftware dafür ein eigenständig lauffähiges und gut dokumentiertes Skript, das die benötigten Werkzeuge inklusive Versionen spezifiziert und lädt (z. B. über das devtools-Paket in R, über requirements.txt in Python) und anschließend die Forschungssoftware Schritt für Schritt ausführt (Stodden & Miguez, 2014; van Atteveldt et al., 2019; Wilson et al., 2017). Im Fall einer eigenständigen Anwendung lässt sich für eine autonome Replikation die genaue Laufzeitumgebung etwa über eine Containervirtualisierung (z. B. Docker) oder entsprechende Konfigurationsdateien (z. B. Makefile, requirements.txt) spezifizieren oder gar emulieren (Hasselbring et al., 2020; van Atteveldt et al., 2019).

Diese Bemühungen um idealtypische Forschungssoftware kommen derzeit allerdings auch in ihren Ursprungsdisziplinen, etwa der Informatik, nur bedingt zu vollem Einsatz. Vielmehr zeichnen sich vielerorts Arbeitsgruppen ab, um fachspezifische Empfehlungen für eine professionellere Forschungssoftware zu diskutieren (z. B. die DGPuK-Arbeitsgruppen für Forschungsdaten und Forschungssoftware). Zentrale Themen sind dabei die direkt ausführbare Archivierung von Forschungssoftware, auch über Versionen und Plattformen hinweg (z. B. im Rahmen der Gesellschaft für Forschungssoftware, in der sich insbesondere Entwickelnde für Forschungsgruppen engagieren; vgl. de-rse.org), die automatisierte Prüfung von Forschungssoftware auf Funktionstüchtigkeit im Vorfeld einer Publikation (vgl. Crouch, 2020) oder die Präregistrierung zur Begutachtung von Forschungssoftware vor Durchführung einer Studie (vgl. Bowman & Keene, 2018; Dienlin et al., 2020; Lewis, 2020).

1.3 Kriterien der Validierbarkeit

Forschungssoftware in der KMW ist validierbar, wenn sie strukturiert und verifiziert ist.

Strukturierte Forschungssoftware ist für andere lesbar und ermöglicht ein schnelles Durchdringen des Quellcodes. Strukturieren lässt sich Software dabei einerseits inhaltlich anhand der zu erfüllenden Aufgaben (z. B. Kommunikation mit API-Schnittstelle, Vorverarbeitung von Textdaten, Auswertung), andererseits technologisch anhand der auszufüllenden Funktionen (z. B. Authentifizierung, Lemmatisierung, Bootstrapping). Um beiden Aspekten bei adäquater Lesbarkeit gerecht zu werden, soll Forschungssoftware nach Aufgaben modularisiert und innerhalb der Module nach Funktionen organisiert sein (Storer, 2017). R ermöglicht eine solche Modularisierung mithilfe von Verzeichnissen, Dateien und dem source-Befehl oder durch das Auslagern in eigene Pakete; eine Organisation innerhalb einzelner Dateien ist über Code-Sektionen, gekennzeichnet etwa mithilfe von Markdown, möglich. Python erlaubt eine Modularisierung über Verzeichnisse (in Python als „Packages“ bezeichnet), Dateien („Modules“) und die import-Befehlspalette; die Organisation innerhalb eines Moduls ist mithilfe von Klassen möglich. Eine solche Code-Struktur erlaubt ferner die konsequente Vermeidung von Redundanzen – nicht zuletzt, um Fehlerquellen zu reduzieren und die Lesbarkeit zu verbessern. Dafür gilt es auch, etablierte Bibliotheken und Pakete Dritter einzusetzen, Du-

plikate im Code durch wiederverwendbare Funktionen zu ersetzen und einzelne Funktionen in zentrale Module auszulagern (Wilson et al., 2017).

Strukturierter Quellcode bietet schließlich die besten Voraussetzungen, um angemessen *verifiziert* zu werden, was nicht nur das manuelle Prüfen auf Plausibilität meint, sondern insbesondere das automatisierte und iterative Testen. Tests lassen sich dabei im Wesentlichen einteilen in „unit tests", „integration tests" und „acceptance tests" (Stodden & Miguez, 2014). Ziel von „unit tests" ist es, einzelne Funktionen und Module auf ihre Sachrichtigkeit zu prüfen, mithin also die Eingabeparameter zu variieren und die Rückgabewerte mit erwarteten Resultaten zu vergleichen. Demgegenüber stehen „integration tests", die das Zusammenspiel von Modulen in den Blick nehmen, also etwa das Scraping mit der anschließenden Verarbeitung der gesammelten Daten. Zuletzt werden insbesondere für Anwendungen immer häufiger „acceptance tests" durchgeführt, bei denen Anwendende oder Auftraggebende die Forschungssoftware testen und anschließend den Entwickelnden Bericht erstatten. Da „unit tests" und „integration tests" softwaregesteuert ablaufen, ist eine Vielzahl an Testumgebungen verfügbar, die sich meist nahtlos in integrierte Entwicklungsumgebungen einfügen (z. B. testthat für R in RStudio, pytest für Python in PyCharm).

Die Entwicklung von Testszenarien ist dabei kaum zu unterschätzen, und ein permanenter Rollenwechsel zwischen kreativ entwickelnd und streng testend der Code-Qualität äußerst zuträglich (Storer, 2017). Die Entwicklung von Tests sollte dabei selbst den diskutierten Kriterien professioneller Forschungssoftware unterliegen, Testszenarien also selbst etwa verständlich und nachvollziehbar sein. Bei ausreichend Ressourcen wird zudem zu fortlaufender gegenseitiger Begutachtung von Code (Kelly & Sanders, 2008), institutionalisiert etwa im Rahmen sogenannter „Merge Requests"/„Pull Requests" (vgl. auch den Abschnitt 1.4 zu Kriterien der Nachhaltigkeit), oder zum „pair programming", also dem Entwickeln zu zweit vor einem Rechner, geraten (Brown & Wilson, 2018); auch Verfahren des lauten Denkens im Rahmen von Tests durch unbeteiligte Dritte vermögen die Qualität von Forschungssoftware zu verbessern. Insbesondere letztere Empfehlungen zielen dabei auch darauf ab, Fehler als konstruktiven Teil des Entwicklungsprozesses zu verstehen und einer solchen Fehlerkultur mit einer angemessenen Testkultur zu begegnen (Kelly & Sanders, 2008; Wilson et al., 2014).

1.4 Kriterien der Nachhaltigkeit

Forschungssoftware in der KMW ist nachhaltig, wenn sie umsichtig und nutzbar ist.

Umsichtig meint dabei die möglichst weitreichende Beachtung äußerer Umstände und verweist als solche zunächst auf die Implementierung von Fehler- und Ausnahmebehandlung (Gentzkow & Shapiro, 2014). Mögliche erwartbare Fehlerquellen (z. B. das Erreichen eines API-Limits) sollen im Code abgefangen und verarbeitet werden, bei unerwarteten Ausnahmen sollen aktuelle Zwischenstände gespeichert und mögliche Ursachen protokolliert werden. Dafür bietet sich die Ausnahmebehandlung einzelner Programmiersprachen an (z. B. tryCatch in R, try/except in Python). Umsichtig meint ferner die Berücksichtigung zentraler Aspekte der Sicherheit und Forschungsethik, etwa im Umgang mit personenbezogenen Daten (Zook et al., 2017). Dabei bieten sich Technologien der Verschlüsselung und Verschleierung, der gesicherten Übertragung sowie der Anonymisierung an. Für den forschungsethischen Umgang im Rahmen von Forschungssoftware wird zudem häufig auf Ethik-Kommissionen (Institutional Review Boards, IRB) sowie den innerdisziplinären Austausch verwiesen (Zook et al., 2017) – beiden Wegen gemein ist dabei der Blick Dritter auf den Code und den Umgang mit Daten.

Tabelle 1: Gütekriterien professioneller Forschungssoftware

Kriterien der Transparenz	
zugänglich	– in etablierten Repositorien veröffentlicht – im Fachdiskurs thematisiert – Code, Daten, Dokumentation, Materialien, Analysen und Beispiele publiziert (im Zweifel anonymisiert)
anschlussfähig	– baut auf verbreitete Programmiersprachen auf (z. B. Python, R) – setzt auf etablierte Bibliotheken und Pakete – berücksichtigt Normen und Standards in Code und Dokumentation (Code-Gestaltung, Code-Struktur, sichergestellt durch moderne IDE)
verständlich	– dokumentiert sämtliche Daten, Variablen und Ausprägungen – besteht aus selbsterklärendem Code (sinnvolle und sprechende Variablen- und Funktionsnamen, rigide Formatierung) – dokumentiert Zweck, Parameter und Rückgabe je Skript und Funktion – dokumentiert in gängiger Auszeichnung (z. B. roxygen2, docstrings)
Kriterien der Replizierbarkeit	
nachvollziehbar	– lückenlos dokumentiert – mitsamt nötigen Rohdaten und Ressourcen veröffentlicht – auf eingesetzte Pakete und Versionsnummern verwiesen
autonom	– lauffähiges und dokumentiertes Skript zur eigenständigen Ausführung der Forschungssoftware veröffentlicht – Laufzeitumgebung möglichst akkurat archiviert
Kriterien der Validierbarkeit	
strukturiert	– nach Aufgaben modularisiert – innerhalb von Modulen nach Funktionen organisiert – Code-Redundanzen entfernt
verifiziert	– Funktionen im Rahmen von „unit tests“ kleinteilig verifiziert – Module im Rahmen von „integration tests“ großflächig getestet – Verifikationsszenarien mithilfe von Testumgebungen implementiert – Dritte über Code-Begutachtung, paarweises Programmieren, lautes Denken im Rahmen der Testszenarien und/oder „acceptance tests“ berücksichtigt
Kriterien der Nachhaltigkeit	
umsichtig	– Fehlerbehandlung für erwartbare Szenarien implementiert – Fehlerbehandlung für Absturz der Forschungssoftware implementiert (Zwischenstände werden gespeichert, Umstände protokolliert) – zentrale Sicherheitsmaßnahmen berücksichtigt – Forschungssoftware auf ethische Aspekte geprüft (z. B. durch IRB)
nutzbar	– lizenziert (im Einklang mit genutzten Bibliotheken und Paketen) – versioniert (z. B. Git, Subversion) – Absichten und Pläne zu längerfristiger Wartung kommuniziert

Um Forschungssoftware nachhaltig *nutzbar* zu machen, ist neben den bereits genannten Kriterien, etwa der Zugänglichkeit oder der Verständlichkeit, eine entsprechende Lizenzierung notwendig. Dabei gilt es, verwendete Bibliotheken und Pakete Dritter auf ihre Lizenz hin zu prüfen, um sich nicht über deren Prämissen hinwegzusetzen, und gegebenenfalls durch offenere Alternativen zu ersetzen (Stodden & Miguez, 2014). Darauf aufbauend sollte Forschungssoftware die offenste unter den möglichen Lizenzen erhalten (z. B. Apache, GNU, MIT) – einerseits, um für Klarheit im Rahmen der Nut-

zung, Weiterentwicklung und Weiterverbreitung zu sorgen; andererseits, um die Nutzung, Weiterentwicklung und Weiterverbreitung auch zu ermöglichen (vgl. Haim & Zamith, 2019). Eine entsprechend wünschenswerte Weiterentwicklung profitiert dabei wiederum von einer umfassenden Dokumentation sowie der konsequenten Nutzung einer kommentierten Versionsverwaltung (Stodden & Miguez, 2014). Moderne Versionsverwaltungssysteme (z. B. Git, Subversion) erlauben auch die Institutionalisierung gegenseitiger Code-Begutachtung („Merge Requests"/„Pull Requests") sowie die Integration sogenannter „Pipelines", die bei Code-Änderungen automatisiert die Durchführung vorab definierter Tests oder Formatierungsschritte ermöglichen. Zuletzt sollte für eine nutzbare Forschungssoftware klar kommuniziert werden, ob und inwiefern die Forschungssoftware aktiv gewartet wird; denn während Industriestandards konsistente Wartung als Qualitätsmerkmal einer Software betrachten (z. B. in der Spezifizierung nach ISO/IEC/IEEE 12207:2017), ist die langfristige Finanzierung der Wartung von Forschungssoftware nur selten Gegenstand von Drittmitteln.

2. Handlungsempfehlungen

Um die Gütekriterien in Handlungsempfehlungen überführen zu können, ist zunächst erneut festzuhalten, dass die Entwicklung von Forschungssoftware meist im Rahmen und für den begrenzten Zeitraum einzelner Forschungsprojekte erfolgt und gefördert wird. Langfristige Projekte mit dem Ziel, entwickelte Forschungssoftware der Allgemeinheit zur Verfügung zu stellen und auch zu warten, sind in der deutschen Kommunikations- und Medienwissenschaft (KMW) nur in Ausnahmefällen zu beobachten. Ähnlich wie in anderen Ländern ist eine Veröffentlichung oder Wartung von Forschungssoftware vielmehr auf das Bestreben einzelner Beteiligter zurückzuführen (van Atteveldt et al., 2019).

Mit Blick in die akademische Softwareentwicklung verschiedener Disziplinen (Storer, 2017) bewegt sich die KMW damit in der Kategorie (1) unprofessioneller Skriptentwicklung, etwa durch (autodidaktisch) programmierende Forschende. Während diese Kategorie größtmögliche kreative Flexibilität erlaubt, ist sie gleichzeitig auf das Wissen und die Motivation Einzelner angewiesen. Auch bekanntere Beispiele in der KMW, etwa der „Facepager" (Jünger & Keyling, 2017), „SoSci Survey" (Leiner, 2014) oder das „tidycomm"-Paket (Unkel, 2019), sind unmittelbar mit ihren Entwicklern verknüpft. Eine solche Form der Entwicklung vermag also gerade in einer frühen Phase Forschungssoftware voranzubringen, sie eignet sich aber nur bedingt für die nachhaltige Institutionalisierung entwickelter Werkzeuge. Um dieser Kreativität nicht nur weiterhin Raum zu geben, sondern sie auch stärker zu fördern, lassen sich vermehrt Aspekte und Gütekriterien professioneller Forschungssoftwareentwicklung in die Ausbildung des wissenschaftlichen Nachwuchses integrieren. Ferner bedarf es sowohl einer kurz- und mittelfristigen extrinsischen Motivation als auch der langfristigen Perspektive, sich vermehrt mit der Entwicklung von Forschungssoftware auseinanderzusetzen.

Im Gegensatz zu Entwicklungen durch Einzelpersonen binden (2) professionelle inneruniversitäre Entwicklungen explizit Programmierende in Forschungsgruppen ein, um mehrere Projekte versorgen und Ressourcen wiederverwenden zu können (Storer, 2017). Die an der London School of Economics and Political Science ansässige und ERC-geförderte Entwicklung und Wartung der unterschiedlichen R-Pakete um das „quanteda"-Projekt zur Analyse von Textdaten (Benoit et al., 2018) fällt beispielsweise in diese Kategorie. Während in der deutschsprachigen KMW kaum entsprechende Beispiele auszumachen sind (eine Ausnahme bilden etwa Hepp et al., 2018), wurde diese Entwicklung im Fach zuletzt über die Schaffung erster Methodenprofessuren mit dezidierter „Com-

putational"-Denomination angestoßen – eine deutliche Incentivierung, deren Auswirkung sich aber erst noch abzeichnen muss. Um eine solche Entwicklung weiter voranzutreiben, ist neben entsprechenden Drittmittelformaten insbesondere eine gewisse Grundausbildung seitens der Antragstellenden nötig.

Zuletzt sieht die (3) institutionalisierte Entwicklung die außeruniversitäre Umsetzung und Wartung kollaborativ im Fach nutzbarer Software-Infrastruktur vor, die lediglich wissenschaftlich initiiert, begleitet und genutzt wird (Storer, 2017). Wenngleich ein solches Auslagern einem gewissen Risiko des Kontrollverlusts unterliegt, stellt es für zentrale Infrastruktur-Anforderungen eines Fachs eine durchaus lohnenswerte Alternative zur fachinternen Entwicklung dar. Beispielhaft ist hier etwa „Media Cloud" als Archiv von Online-Nachrichten zu nennen, das durch seine US-Förderung aber den Fokus zunächst auf englischsprachige Nachrichten legt; vergleichbare Infrastruktur-Förderungen in der deutschsprachigen KMW sind nicht bekannt. Auch hierfür wären neben einer gewissen Grundausbildung seitens der Antragstellenden insbesondere entsprechende Drittmittelformate für die Projektplanung erforderlich.

2.1 Methodenausbildung stärken

Die Methodenausbildung wie auch die Anwendung empirischer Methoden gehört in der KMW zu allen gängigen Curricula (Matthes, 2019; Matthes et al., 2011). Innerhalb dieser wie auch im Rahmen von Abschlussarbeiten können Gütekriterien professioneller Forschungssoftware Platz finden, um so bereits in Bachelor- und Masterstudiengängen Grundsteine für spätere Arbeiten zu legen. Einen niederschwelligen Einstieg in eine professionellere Entwicklung von Forschungssoftware ermöglicht dabei die Datenauswertung, zumal alle gängigen Programme der Datenauswertung den Fokus auf Skripte erlauben. Für die Vermittlung von Gütekriterien lassen sich die Skripte selbst zunächst zu einer Prüfungsleistung in Veranstaltungen der Datenanalyse oder der Statistik erklären. Diese Skripte sollen, in den Gütekriterien professioneller Forschungssoftware gesprochen, anschlussfähig, verständlich, nachvollziehbar, strukturiert und verifiziert sein. Studierende sind so frühzeitig dazu angehalten, ordentlichen und selbsterklärenden Code zu schreiben, ausführlich zu dokumentieren, zu modularisieren, zu strukturieren und zu testen. Darauf aufbauend können solche Skripte unter den Studierenden getauscht und einer gegenseitigen Begutachtung unterzogen werden.

Ferner lassen sich Kriterien einer zugänglichen, anschlussfähigen, umsichtigen und nutzbaren Forschungssoftware in Forschungsseminare einbetten. Hierbei kann etwa auf bestehende Daten und Software aufgebaut werden, beispielsweise im Rahmen von Replikationsstudien, die Studierende zur detaillierten Auseinandersetzung mit zugänglichen Ressourcen zwingen. Darüber hinaus können sowohl die längerfristige Nutzbarmachung der eigenen Forschungssoftware sowie die Einbindung fremder Bibliotheken und Pakete zu zentralen Herausforderungen der Datenerhebung oder Auswertung gemacht werden, um ein Bewusstsein für die Anschlussfähigkeit von Forschungssoftware zu schaffen. Es bietet sich außerdem an, Fragen der Fehler- und Ausnahmebehandlung sowie Aspekte der Datensicherheit und Forschungsethik im Rahmen eines umsichtigen Forschungsprozesses zu thematisieren.

Auch in Abschlussarbeiten kann die Anwendung der Gütekriterien professioneller Forschungssoftware selbstverständlich(er) werden, indem etwa Skripte der Datenauswertung anschlussfähig, nachvollziehbar, strukturiert und verifiziert, die Daten entsprechend zugänglich und verständlich sein sollen. Ferner kann auch die Entwicklung neuer, die Validierung bestehender oder der Vergleich unterschiedlicher Forschungssoftware bisweilen zum zentralen Gegenstand erhoben werden. Hierbei ließen sich ne-

ben den genannten Gütekriterien auch Aspekte der Anwendungsfreundlichkeit berücksichtigen. Eine solche Ausdifferenzierung käme wohl auch den zweifellos unterschiedlichen Zielgruppen unter den Studierenden entgegen. So ist neben einer wohl auch hier zu erwartenden Gruppe (autodidaktisch) Programmierender sicherlich auch von einer Gruppe tiefergehend Interessierter auszugehen, denen es zwar an konkreteren Programmierkenntnissen, jedoch nicht an Motivation mangelt, sowie von einer Gruppe eher oberflächlich Interessierter, die Forschungssoftware zwar nutzen, aber vermutlich nicht selbst entwickeln würden.

2.2 Extrinsische Motivation fördern

Über die Ausbildung hinaus ist neben dem Wissen über professionelle Forschungssoftware wohl insbesondere die kurz- und mittelfristige extrinsische Motivation, sich damit zu beschäftigen, entscheidend. Ein maßgeblicher Hebel dafür ist die institutionalisierte Anerkennung, die sich im Wissenschaftssystem insbesondere in Zitationen bemisst. Sie erfolgt allem voran über Fachzeitschriften, die dadurch auch in die Lage versetzt werden, professionelle Standards zu setzen, zu fordern und zu fördern.

Dabei scheint für Zeitschriften in unserem Fach Innovation im Umgang mit Forschungssoftware derzeit insbesondere zu bedeuten, Publikationen, die Open-Science-Standards der Transparenz von Daten oder der Präregistrierung folgen, mit sogenannten „Badges“ hervorzuheben (Blohowiak et al., 2013). Solche Bemühungen sind lobenswert und erfordern gleichzeitig minimalen Aufwand in ihrer Implementierung. Sie bergen aber die Gefahr, unterschiedliche Motive miteinander zu vergleichen – ist die präregistrierte Befragung mit Badge wertvoller als die Inhaltsanalyse oder der Theoriebeitrag ohne Badge?

Ergänzend wäre deshalb ein fundierterer Umgang mit Forschungssoftware wünschenswert, der nicht nur die Transparenz von Forschungssoftware über Badges lobt, sondern sie schlicht voraussetzt. Derartige Forderungen können von Fachzeitschriften selbst erhoben, sie können aber auch im Begutachtungsprozess eingefordert werden. Auch die Validierbarkeit von Forschungssoftware könnte unter bestimmten Voraussetzungen, etwa wenn neue Forschungssoftware zum Einsatz kommt, zu einem zentralen Kriterium werden, das es ebenfalls zu begutachten gilt. Ein solches System wäre gleichzeitig einer nachhaltigeren Landschaft an Forschungssoftware zuträglich, was gleichsam Entwickelnden messbare akademische Anerkennung (Zitationen) einbrächte. Die notwendige Begutachtung von Code würde ferner, entsprechende Anforderungen an den Code vorausgesetzt, der Anschlussfähigkeit und Verständlichkeit der Forschungssoftware entgegenkommen.

Im Umkehrschluss bedeutet ein solches System, dass es begutachteter Möglichkeiten der Publikation von Forschungssoftware bedarf. Diese bestehen bisweilen bereits in den bekannten Fachzeitschriften mit methodischem Schwerpunkt (z. B. Computational Communication Research, Communication Methods and Measures), sie unterliegen aber den im Fach üblichen Charakteristika wissenschaftlicher Publikationen. Damit bedeuten sie in ihrer aktuellen Form zusätzlichen Aufwand für die Entwickelnden. Im Sinne einer professionelleren Forschungssoftwarelandschaft wären indes auch Publikationsformen nötig, die etwa Anleitungen enthalten (z. B. mit Videos), Tests erlauben (z. B. über Pipelines) oder Laufzeitumgebungen spezifizieren (z. B. über requirements.txt). Selbst die Implementierung von Versionen wäre denkbar, um etwa das nachträgliche Beheben kleinerer Fehler, nicht aber das nachträgliche Integrieren neuer Funktionen, im Rahmen einer Publikation zu ermöglichen.

2.3 Drittmittel ermöglichen

Im weiteren Feld kurz- und mittelfristiger extrinsischer Motivation liegen sicherlich auch Drittmittel. Sie sollen im Idealfall eine Möglichkeit bieten, sich durch die Entwicklung und Wartung einer angemessen der Forschungsgemeinschaft zur Verfügung gestellten Forschungssoftware verdient zu machen. Dafür bedarf es insbesondere entsprechender Förderformate, wie sie bislang nur gelegentlich in Einzelaufrufen formuliert werden (z. B. das „IDEAS"-Programm des ERC). Sie können einerseits entwickelnde Forschende ermutigen, mit Forschungssoftware lohnend zum Diskurs beizutragen; andererseits erhöht eine solch dezidierte Förderung mittelfristig wohl auch die Güte jener Forschung, die von so geförderter Forschungssoftware profitiert.

Letzteres lässt sich etwa mit Storer (2017) zeigen, der in einem systematischen Überblick über Fallstudien akademischer Softwareentwicklung feststellt, dass insbesondere fehlende Informationen (Transparenz), eine nur lückenhaft nachvollziehbare Anwendung (Replizierbarkeit) sowie eine nicht (Validierbarkeit) oder nicht mehr (Nachhaltigkeit) intakte Funktionalität die Neuentwicklung von originär bereits bestehender Forschungssoftware notwendig machen. Andere, stärker inhaltlich fokussierte Drittmittelanträge sind entsprechend immer wieder gezwungen, die Entwicklung notwendiger Forschungssoftware selbst vorzusehen. Ein mühsames und kostspieliges Unterfangen, das aufgrund von projektspezifischen Perspektiven und Rahmenbedingungen auch einer Vergleichbarkeit im Weg steht.

Dieser Punkt lässt sich erneut am Beispiel „quanteda" (Benoit et al., 2018) verdeutlichen: Quantitative Inhaltsanalysen sind Teil vieler Forschungsprojekte in der KMW, die zunehmend wachsenden Textmengen machen eine zumindest teilweise Automatisierung der Auswertung nötig; eine Entwicklung der Forschungssoftware zu einer solchen Auswertung ist aufwendig und bedarf zahlreicher methodischer Entscheidungen (Günther & Quandt, 2016) – Entscheidungen, die das in diesem Fall über acht Jahre ERC-geförderte Projekt in einem transparenten Prozess bereits getroffen hat. Und obschon es in einzelnen Forschungsprojekten Gründe für oder gegen einzelne solcher Entscheidungen geben mag, etwa wie Begriffe auf ihre Wortstämme zu reduzieren oder die Dokumententropie zu berechnen sind, so schafft eine derart verbreitete Forschungssoftware wie „quanteda" Normen und Standards für solche Verfahren und damit eine Infrastruktur für einen methodischen Konsens.

2.4 Perspektive schaffen

Über Zitationen und Drittmittel hinaus bedarf es zuletzt einer langfristigen Perspektive, um sich im Fach mit Forschungssoftware zu beschäftigen. Diese Perspektiven sind zuletzt im Zuge einer „Computational Communication Science" sowie einer „Computational Social Science" stark gewachsen (Lazer et al., 2020; van Atteveldt et al., 2019): Entsprechende Fachgruppen haben sich herausgebildet oder ausgerichtet (z. B. Methoden-Fachgruppe in der DGPuK, „Computational Methods"-Interessensgruppe in der ICA), Arbeitsgruppen und Diskussionsrunden sind allerorts anzutreffen (z. B. waren allein im ersten Quartal des Jahres 2020 innerhalb der DGPuK mehrere entsprechende Initiativen aktiv, etwa zum Umgang mit Forschungsdaten, Forschungssoftware und Computational Communication Science in der Lehre), Fachzeitschriften mit originärem Methodenbezug sind zentraler Bestandteil der Fachliteratur (z. B. Communication Methods and Measures, Computational Communication Research), Ausschreibungen für Drittmittel mit – wenngleich primär inhaltlichem – „Computational"-Bezug in der KMW immer wieder verfügbar (z. B. KI-Initiative der DFG, Computational-Social-

Science-Bemühungen der Volkswagenstiftung, AI-Impact-Challenge von Google). Mit dieser Dynamik einher geht einerseits eine höhere Sichtbarkeit, andererseits wohl auch ein erhöhter Bedarf entsprechender Kompetenz und Erfahrung (vgl. die anderen drei Handlungsempfehlungen).

Damit zeigt sich in Grundzügen auch eine zentralere Verschiebung im Feld: Herrschte bislang die Meinung vor, die eigene wissenschaftliche Karriere nicht durch Methoden voranbringen zu können (Matthes, 2019), so scheinen Computational Methods einen Grundstein einer stärker institutionalisierten Perspektive zu legen. So haben inzwischen einige wenige Universitäten entsprechende (Junior-)Professuren der Computational Methods implementiert und geben so der professionellen Beschäftigung mit Methoden – und damit zu einem ganz wesentlichen Teil mit Forschungssoftware – einen institutionellen Rahmen. Sich innerhalb der KMW professionell mit Forschungssoftware auseinanderzusetzen, entwickelt sich langsam zur Karriereoption.

3. Fazit

Eine ständige Entwicklung digitaler Medien- und Informationsumgebungen macht die fortlaufende Anpassung von Forschungssoftware – innerhalb der Fachgemeinde entwickelte Computerprogramme für jegliche Zwecke entlang des Forschungsprozesses – nötig. Dabei finden Entwicklung und Wartung in der Kommunikations- und Medienwissenschaft (KMW) häufig innerhalb einzelner Forschungsprojekte und oftmals durch (autodidaktisch) programmierende Forschende statt. Ein Umstand, der Innovation und Kreativität ermöglicht, gleichzeitig aber keiner nachhaltigen Institutionalisierung entwickelter Werkzeuge zuträglich ist.

Der Beitrag schlägt daher neun Gütekriterien professioneller Forschungssoftware vor, die für sich genommen niederschwellig im Rahmen der Entwicklung umsetzbar sind (Tabelle 1). Forschungssoftware sollte demnach zugänglich, anschlussfähig, verständlich, nachvollziehbar, autonom, strukturiert, verifiziert, umsichtig und nutzbar sein. Diese Gütekriterien ergeben sich aus den vier zentralen Pfeilern der Entwicklung von Forschungssoftware – Transparenz, Replizierbarkeit, Validierbarkeit und Nachhaltigkeit –, die sich in Disziplinen mit fortgeschrittener Diskussion um die Güte von Forschungssoftware (z. B. Informatik, Bioinformatik, informatischer Maschinenbau, informatische Medizin) abgezeichnet haben.

Für eine längerfristige Qualitätssicherung der Forschungssoftware in der KMW ist neben diesen Gütekriterien indes wohl eine Veränderung einiger Rahmenbedingungen nötig. Der Beitrag schlägt dafür vier konkrete Handlungsempfehlungen vor:

(1) Möglichst früher Kontakt der Studierenden mit der Entwicklung professioneller Forschungssoftware, etwa im Rahmen von Datenanalysekursen, Forschungsseminaren oder Abschlussarbeiten; so sollen Berührungsängste abgebaut und gleichzeitig der Grundstein für einen professionelleren Umgang mit Skripten und Programmierung in der Forschung gelegt werden.
(2) Angemessene Möglichkeiten der Veröffentlichung und Zitation, etwa durch eingeforderte Standards, adäquate Begutachtungsprozesse und flexiblere Publikationsformate; sie sollen Sichtbarkeit, Qualitätsstandards und eine entsprechende Reputation von Entwickelnden sicherstellen.
(3) Förderformate, die das Einwerben von Drittmitteln durch die Entwicklung und Wartung von Forschungssoftware ermöglichen; so soll Entwicklung gefördert und gleichzeitig das Fach stärker mit entsprechender Infrastruktur ausgestattet werden.

(4) Institutionalisierte Anreize, die eine professionelle Auseinandersetzung mit Forschungssoftware auch in Karriereoptionen übersetzen (z. B. Methodenprofessuren).

Empirische Forschung soll durch professionellere Forschungssoftware nicht eingeschränkt werden. Das Gegenteil ist der Fall: Zugängliche und verständliche Werkzeuge, strukturierte, nachvollziehbare und autonome Skripte der Datenauswertung, verifizierte und umsichtige Applikationen sind das methodische Rückgrat einer vertrauenswürdigen empirischen Wissenschaft. Die Güte eingesetzter Forschungssoftware soll kein notwendiges Beiwerk, sondern selbstverständliche Voraussetzung sein. Ist eine entwickelte Forschungssoftware zudem nutzbar und anschlussfähig, ermöglicht sie darüber hinaus eine gegenseitige Bezugnahme, Anerkennung, Weiterentwicklung und Wertschätzung, um mittel- und langfristig nicht nur auf intrinsisch motivierte Einzelpersonen, sondern auch auf eine institutionalisierte Infrastruktur an Forschungssoftware zurückgreifen zu können.

Literatur

Almeida, A. V. de, Borges, M. M., & Roque, L. (2017). The European Open Science Cloud: A new challenge for Europe. *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, 1–4. https://doi.org/10.1145/3144826.3145382.

Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S., & Matsuo, A. (2018). quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software*, *3*(30), 774. https://doi.org/10.21105/joss.00774.

Blohowiak, B. B., Cohoon, J., de-Wit, L., Eich, E., Farach, F. J., Hasselman, F., Holcombe, A. O., Humphreys, M., Lewis, M., & Nosek, B. A. (2013). *Badges to acknowledge open practices*. https://osf.io/tvyxz/.

Bowman, N. D., & Keene, J. R. (2018). A layered framework for considering open science practices. *Communication Research Reports*, *35*(4), 363–372. https://doi.org/10.1080/08824096.2018.1513273.

Brown, N. C. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLOS Computational Biology*, *14*(4), e1006023. https://doi.org/10.1371/journal.pcbi.1006023.

Chan, C., Zeng, J., Wessler, H., Jungblut, M., Welbers, K., Bajjalieh, J., van Atteveldt, W., & Althaus, S. (2020). Reproducible extraction of cross-lingual topics. *Communication Methods & Measures*. https://doi.org/10.1080/19312458.2020.1812555

Crouch, S. (2020, Januar 20). *Software design in just 20 questions*. Software Sustainability Institute. https://software.ac.uk/blog/2020-01-20-software-design-just-20-questions [11.11.2020].

Crouch, S., Hong, N. C., Hettrick, S., Jackson, M., Pawlik, A., Sufi, S., Carr, L., De Roure, D., Goble, C., & Parsons, M. (2013). The software sustainability institute: Changing research software attitudes and practices. *Computing in Science Engineering*, *15*(6), 74–80. https://doi.org/10.1109/MCSE.2013.133.

DFG (2019, Juni 18). *Qualitätssicherung von Forschungssoftware durch ihre nachhaltige Nutzbarmachung*. Deutsche Forschungsgemeinschaft. https://www.dfg.de/foerderung/info_wissenschaft/2019/info_wissenschaft_19_44/index.html

Dienlin, T., Johannes, N., Bowman, N. D., Masur, P. K., Engesser, S., Kümpel, A. S., Lukito, J., Bier, L. M., Zhang, R., Johnson, B. K., Huskey, R., Schneider, F. M., Breuer, J., Parry, D. A., Vermeulen, I., Fisher, J. T., Banks, J., Weber, R., Ellis, D. A., … de Vreese, C. H. (2020). An agenda for open science in communication. *Journal of Communication*. https://doi.org/10.1093/joc/jqz052.

Freelon, D. (2018). Computational research in the post-API age. *Political Communication*, *35*(4), 665–668. https://doi.org/10.1080/10584609.2018.1477506.

Gentzkow, M., & Shapiro, J. M. (2014). *Code and data for the social sciences: A practitioner's guide*. University of Chicago Press. http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf [11.11.2020].

Günther, E., & Quandt, T. (2016). Word counts and topic models. Automated text analysis methods for digital journalism research. *Digital Journalism*, *4*(1), 75–88. https://doi.org/10.1080/21670811.2015.1093270.

Haim, M. (2018). Mehr methodischer Mut, bitte! *Aviso*, *66*, 8.

Haim, M. (2020). Agent-based testing: An automated approach toward artificial reactions to human behavior. *Journalism Studies*, *21*(7), 895–911. https://doi.org/10.1080/1461670X.2019.1702892.

Haim, M., & Zamith, R. (2019). Open-source trading zones and boundary objects: Examining GitHub as a space for collaborating on „news". *Media and Communication*, *7*(4), 80. https://doi.org/10.17645/mac.v7i4.2249.

Hasselbring, W., Carr, L., Hettrick, S., Packer, H., & Tiropanis, T. (2020). From FAIR research data toward FAIR and open research software. *It – Information Technology*, *62*(1), 39–47. https://doi.org/10.1515/itit-2019-0040.

Hepp, A. (2016). Kommunikations- und Medienwissenschaft in datengetriebenen Zeiten. *Publizistik*, *61*(3), 225–246. https://doi.org/10.1007/s11616-016-0263-y.

Hepp, A., Breiter, A., Hasebrink, U., & Loosen, W. (2018). *Me-Software: Software for qualitative media and communication research*. https://mesoftware.org/ [11.11.2020].

Jünger, J., & Keyling, T. (2017). *Facepager. An application for generic data retrieval through APIs*. https://github.com/strohne/Facepager/ [11.11.2020].

Kelly, D., & Sanders, R. (2008, Mai). *Assessing the quality of scientific software*. First International Workshop on Software Engineering for Computational Science & Engineering, Leipzig. https://se4science.org/workshops/secse08/Papers/Kelly.pdf [11.01.2021].

Lazer, D. M. J., Pentland, A., Watts, D. J., Aral, S., Athey, S., Contractor, N., Freelon, D., Gonzalez-Bailon, S., King, G., Margetts, H., Nelson, A., Salganik, M. J., Strohmaier, M., Vespignani, A., & Wagner, C. (2020). Computational social science: Obstacles and opportunities. *Science*, *369*(6507), 1060–1062. https://doi.org/10.1126/science.aaz8170.

Leiner, D. (2014). *Convenience samples from online respondent pools: A case study of the SoSci Panel* [Working paper]. https://www.researchgate.net/publication/259669050 [11.11.2020].

Lewis, N. A. Jr. (2020). Open communication science: A primer on why and some recommendations for how. *Communication Methods and Measures*, *14*(2), 71–82. https://doi.org/10.1080/19312458.2019.1685660.

Matthes, J. (2019). Viel Luft nach oben. Eine kritische Reflexion zum Stellenwert der Methoden in der Kommunikationswissenschaft. In H. Schramm, J. Matthes, & C. Schemer (Hrsg.), *Emotions Meet Cognitions. Zum Zusammenspiel von emotionalen und kognitiven Prozessen in der Medienrezeptions- und Medienwirkungsforschung* (S. 93–103). Springer. https://link.springer.com/chapter/10.1007/978-3-658-25963-1_8 [11.11.2020].

Matthes, J., Kuhlmann, C., Gehrau, V., Jandura, O., Möhring, W., Vogelgesang, J., & Wünsch, C. (2011). Zur Methodenausbildung in kommunikationswissenschaftlichen Bachelor- und Masterstudiengängen. *Publizistik*, *56*(4), 461–481. https://doi.org/10.1007/s11616-011-0133-6.

Mons, B., Neylon, C., Velterop, J., Dumontier, M., da Silva Santos, L. O. B., & Wilkinson, M. D. (2017). Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. *Information Services & Use*, *37*(1), 49–56. https://doi.org/10.3233/ISU-170824.

Nosek, B. A., Alter, G., Banks, G. C., Borsboom, D., Bowman, S. D., Breckler, S. J., Buck, S., Chambers, C. D., Chin, G., Christensen, G., Contestabile, M., Dafoe, A., Eich, E., Freese, J., Glennerster, R., Goroff, D., Green, D. P., Hesse, B., Humphreys, M., … Yarkoni, T. (2015). Promoting an open research culture. *Science*, *348*(6242), 1422–1425. https://doi.org/10.1126/science.aab2374.

Rat für Sozial- und Wirtschaftsdaten (2019). Big Data in den Sozial-, Verhaltens- und Wirtschaftswissenschaften: Datenzugang und Forschungsdatenmanagement. *RatSWD Output*, *4*(6). https://doi.org/10.17620/02671.39.

Stodden, V., & Miguez, S. (2014). Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software*, *2*(1), 1–6. https://doi.org/10.5334/jors.ay.

Storer, T. (2017). Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Computing Surveys*, *50*(4), 1–32. https://doi.org/10.1145/3084225.

Unkel, J. (2019). *tidycomm: Data Modification and Analysis for Communication Research*. https://github.com/joon-e/tidycomm [11.11.2020].

Unkel, J. (2020). *Computational Methods in der politischen Kommunikationsforschung*. https://bookdown.org/joone/ComputationalMethods/ [11.11.2020].

van Atteveldt, W., & Peng, T.-Q. (2018). When communication meets computation: Opportunities, challenges, and pitfalls in computational communication science. *Communication Methods and Measures*, *12*(2–3), 81–92. https://doi.org/10.1080/19312458.2018.1458084.

van Atteveldt, W., Strycharz, J., Trilling, D., & Welbers, K. (2019). Toward open computational communication science: A practical road map for reusable data and code. *International Journal of Communication*, *19*, 3935–3954.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T., Miller, E., Bache, S., Müller, K., Ooms, J., Robinson, D., Seidel, D., Spinu, V., … Yutani, H. (2019). Welcome to the Tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686.

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P., & Wilson, P. (2014). Best practices for scientific computing. *PLoS Biology*, *12*(1), e1001745. https://doi.org/10.1371/journal.pbio.1001745.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, *13*(6), e1005510. https://doi.org/10.1371/journal.pcbi.1005510.

Zook, M., Barocas, S., boyd, D. M., Crawford, K., Keller, E., Gangadharan, S. P., Goodman, A., Hollander, R., Koenig, B. A., Metcalf, J., Narayanan, A., Nelson, A., & Pasquale, F. (2017). Ten simple rules for responsible big data research. *PLOS Computational Biology*, *13*(3), e1005399. https://doi.org/10.1371/journal.pcbi.1005399.