"I really don't like Gaussian processes. I wanna write Python libraries." — A Mixed Methods Investigation of Research Software Engineering in the U.K.

Graham Lee

Kellogg College
University of Oxford

A thesis submitted for the degree of Doctor of Philosophy

Michaelmas 2024

Abstract

Software and its construction are of critical importance to an increasing amount of modern research. Many researchers and creators of research software point to problems in the research community's engagement with software and the work that goes into it, including failing to recognise software authors as research contributors, and limited software training in research.

From its introduction in the UK research software community approximately a decade ago, the role of Research Software Engineer (RSE) has gained adoption. An RSE may be a skilled software engineer who focuses on the construction of software for use in research, or a researcher who specialises in creating software to support their research goals. In a brief time, RSE has gone from a defining term created at a workshop on software sustainability to a job title and group name used at numerous research institutions worldwide, with member societies sharing experience and advocating for career recognition. In short, we are witnessing the creation of a new profession of Research Software Engineering.

In this thesis, we review the literature on RSE and related fields, and identify that no systematic exploration of the role or the developing profession, nor theoretical analysis, has yet been undertaken. We design a mixed-methods methodology to address this gap, drawing on relevant theories from the sociology of professions to investigate RSE as understood and practised in the UK.

We implement one quantitative and three qualitative studies of the field, addressing questions on: the state of software engineering knowledge in research; the state of RSE as a profession; the independence of RSE from existing research practice; and the connection between the relationship of researchers and RSEs, and research software practice. Combining these studies into a single mixed-methods programme, we characterise the state of RSE one decade since its inception, identify gaps in how the role supports sustainable software in research, and provide an evidence-based "manifesto" for future change in the field.

“I really don’t like Gaussian processes. I wanna write Python libraries.” — A Mixed Methods Investigation of Research Software Engineering in the U.K.

Graham Lee

Kellogg College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2024

This thesis is dedicated to the memory of
Leonidas, King of the Spartans
in gratitude for the many mice he brought me

Acknowledgements

Primarily, my thanks go to my supervisors: Professor David Gavaghan; Professor Susanna-Assunta Sansone; and Doctor Helena Webb; for their support, encouragement, expertise, and guidance throughout this research. Each of them demonstrated an enthusiasm for the project that made it delightful in the high times, and worth persevering with in the low times.

Colleagues throughout the Department of Computer Science, the Oxford e-Research Centre, the Oxford Research Software Engineering group, Kellogg College, and the wider university also offered support and advice where it helped, and social activities where distraction was needed. I particularly want to thank the members of the Data Readiness Group who agreed to be the test cohort for my focus group research, and Doctor Martin Robinson for sharing his survey on research software needs.

Speaking of colleagues, my workmates, managers, conference co-attendees, and mentors throughout my career in both industry and academia have taught me so much about writing software, and about thinking about writing software, and all of that was a necessary part of even making this thesis conceivable. In other words, if I have coded far, it is by typing on the shoulders of giants.

Research Software Engineering is a diverse community of people who come together through a shared interest in supporting research through quality software. Many members of the Society for RSE participated in my research; thank you to them and to everybody else who participated. But additionally, thank you to everyone who took a personal interest in my work, discussing it at RSE conferences, collaborations workshops, on podcasts, and online.

I typeset this thesis in LaTeX and thank Leslie Lamport and Donald Knuth for making it easy to manage a document of this size.

Finally, pursuing a doctorate is a significant personal investment, that takes time and focus away from other aspects of a student's life. Special thanks and love are due to Tina, who supported me, encouraged me, and accepted when I spent days at a time in the Department or in my study at home. Thank you also to the close friends who gave me support during this time and listened to my answer to those special words: "how's the thesis going?" And apologies to those other friends who I haven't seen in a long while; hopefully this acknowledgement goes some way to explaining where I've been all this time.

Abstract

Software and its construction are of critical importance to an increasing amount of modern research. Many researchers and creators of research software point to problems in the research community's engagement with software and the work that goes into it, including failing to recognise software authors as research contributors, and limited software training in research.

From its introduction in the UK research software community approximately a decade ago, the role of Research Software Engineer (RSE) has gained adoption. An RSE may be a skilled software engineer who focuses on the construction of software for use in research, or a researcher who specialises in creating software to support their research goals. In a brief time, RSE has gone from a defining term created at a workshop on software sustainability to a job title and group name used at numerous research institutions worldwide, with member societies sharing experience and advocating for career recognition. In short, we are witnessing the creation of a new profession of Research Software Engineering.

In this thesis, we review the literature on RSE and related fields, and identify that no systematic exploration of the role or the developing profession, nor theoretical analysis, has yet been undertaken. We design a mixed-methods methodology to address this gap, drawing on relevant theories from the sociology of professions to investigate RSE as understood and practised in the UK.

We implement one quantitative and three qualitative studies of the field, addressing questions on: the state of software engineering knowledge in research; the state of RSE as a profession; the independence of RSE from existing research practice; and the connection between the relationship of researchers and RSEs, and research software practice. Combining these studies into a single mixed-methods programme, we characterise the state of RSE one decade since its inception, identify gaps in how the role supports sustainable software in research, and provide an evidence-based "manifesto" for future change in the field.

Contents

1 Introduction

Contents

1.1 Overview of the Research Area

In this thesis, we explore the values and practices adopted by Research Software Engineers (RSEs)—an emerging job title proposed in 2012 to increase recognition of the contribution made to research by software specialists and to allow them to create certification and career progression opportunities [1] (see section 2.2). Research Software Engineers work in academic contexts including universities and government research institutions, and focus on constructing and supporting custom software that furthers the goals of their researcher colleagues.

RSE is a simple term that masks a number of complex ideas: the evolving definition of software engineering; the responsibilities of software engineers as perceived by themselves and by other stakeholders in their work; the values

held by software engineers and how these values are expressed in practice; the interaction between how computational researchers and RSEs perceive their own capabilities and those of software engineers in industry; and the requirements and expectations of the various stakeholders in research projects that engage with software and its construction.

We focus specifically on RSE in the United Kingdom, as the title was defined at a national workshop on software sustainability and therefore the role is most established in this country.

1.2 Identifying the Research Gap

We discuss the background of our research in chapter 2, to define the context of RSE including the types of software and the roles people adopt as stakeholders in a research software endeavour. Noting that the creators of the field explicitly connected RSE with software engineering by adopting the phrase in the job title, we situate it in the development of software engineering. We examine crises and debates that characterise the software engineering field, including the software crisis and how Agile Software Development responded to it, calls for professionalising the field or treating it as a craft discipline, and multiple big debates around software ethics including the effects of bias in artificial intelligence systems. We also discuss two questions that relate the ethical and professional debates of RSE to its academic context: Responsible Research and Innovation; and Findable, Accessible, Interoperable, and Reusable research software.

In chapter 3, we review relevant literature to characterise the state of the art and to identify a research gap. We find the field of RSE to be primarily defined by opinion articles that express the practices people wish were universal in research software, opinions that are presented without the support of substantive evidence. We therefore turn to adjacent fields, including the studies of professionalisation and of software engineering, to identify fruitful approaches and techniques that could be applied to RSE.

1.3 Asking Research Questions and Selecting a Methodology

Taking the findings from the literature review into account, in chapter 4 we select and justify a mixed methods approach to researching the field, and present an initial list of hypotheses and research questions. Our hypotheses are:

H1 Researchers working in research institutions in the UK identify a software skills gap.

H2 Although people within the RSE community are trying to define it as a distinct profession, RSE is tightly coupled to practices and expectations in research.

H3 RSE is heavily dependent on the structures and institutions of research, and this limits the opportunity to define distinct outcomes or expectations.

H4 The more a research group "embeds" RSE into its structure, the less opportunity for distinct outcomes and expectations from the group's research software efforts.

Our research questions are:

RQ1 Do UK-based researchers have the skills and training they need to fulfil the software-related requirements of their work?

RQ2 Does RSE exhibit the four properties of a profession in the functionalist theory of professions: a body of expert knowledge; technical autonomy; valuing service to others; and privileged status?

RQ3 How do the institutions and logics of research influence the way an RSE performs their work, and what competing institutions vie for legitimacy?

RQ4 Does the structure of the relationship between researchers and RSEs in a research group affect the research software practice?

We acknowledge that in a lightly-explored field with few existing empirical results in the literature, no single theoretical approach proves compelling as a foundation for our analysis so we choose to apply multiple theoretical models, by selecting a different theoretical basis for each research chapter. We built the mixed-methods study sequentially, using the preliminary analysis and results of each phase to identify areas for further exploration. We subsequently compare the findings from each chapter with the data gathered from other chapters, to seek out congruence or contradiction among the results found using the different theories.

Additionally, we discuss research epistemologies and present our selection of a social constructivist approach to working with participants and analysing the data they create in our research. We explain that this approach would be suitable for developing into a grounded theory of RSE, but that our research should be considered as preliminary exploration and understanding of the field. This work could form the basis of a grounded theory generation, but achieving theoretical saturation in such a broad and unexplored research topic is beyond the scope for a fixed-duration doctoral research programme.

1.4 Undertaking Research

Chapters 5–8 contain the primary research contributions of this thesis.

In chapter 5, we present the design, implementation, and analysis of responses of a survey constructed to understand how important research software is to researchers. We distributed the survey to members of Oxford University. We ask how much time people spend working on research software, how they acquired their research software skills, whether they work with other people on research software, and whether they can afford to engage research software engineers.

In chapter 6, we describe a series of open-ended interviews we undertook with RSEs and their collaborators from various institutions in the UK, in which we asked about aspects of research software engineering as traits in a functionalist model of professions. We also describe online participant councils, to which all interview

participants were invited, where we presented preliminary analysis and invited the participants to discuss the research and its findings.

In chapter 7, we discover and analyse public blog posts written for audiences of research software engineers. We use neo-institutional theory (NIT) for this analysis, seeking evidence of the institutions that drive RSE's approach to work, the logic that these institutions propose in how RSEs do the work, the conflicts that arise when the institutions' interests do not align, and how people act to promote the legitimacy of their preferred institutions.

In chapter 8, we present the design, implementation, and analysis of a series of focus groups we conducted with researchers and RSEs. The participants in each focus group instance are members of the same research group or collaboration, so work together on a research software project, or a research project with a software component. We set each focus group a pair of activities, designed to explore how they work together and how they think about and negotiate various attributes of software engineering. We analyse these data using the theory of communities of practice, which explores how people with a shared interest come together to engage in and improve that practice.

1.5 Analysing Results and Drawing Conclusions

We compare the results from each of our research chapters in chapter 9, to implement the mixed methods approach that we set out in chapter 4. We illuminate the results from our quantitative, survey instrument (see chapter 5) with data from the qualitative studies. Additionally, we search the data from each of the qualitative studies for examples that reinforce or contradict the findings from each of the other studies, to compare the results we gathered using each of the distinct theories.

Finally, in chapter 10, we draw conclusions from our research that can inform decisions made by RSEs, their leaders, and other stakeholders including their funders. We find that the capability of RSE to effect change in the culture of UK academia around research software is limited by its liminal existence in the shadows of academia, meaning that individual practitioners must satisfy the existing

expectations of senior researchers and other stakeholders and perform their software engineering work within that framework. We also find that software engineering knowledge within RSE is incomplete and is inconsistently applied, and that RSE is predominantly active in the sciences with little presence in the social sciences, arts, or humanities.

We present recommendations for stakeholders in research software to change practice based on the conclusions we draw. These include systematically determining what research software needs to be developed in a sustainable fashion; ensuring resources are available for ongoing maintenance of sustained research software; and introducing training that addresses the software engineering skills gap.

We describe further work that can follow from the research presented in this thesis, and present our concluding thoughts on the research.

2

Background

Contents

2.1 Introduction

In chapter 1, we outlined a programme of research into the values and practices of Research Software Engineering (RSE). Here we present some background information on RSE, its development, and on the related fields it specialises or borrows from, as context for our research.

We review the development of software engineering as a profession, including understanding of its responsibilities to society and the arguments for and challenges against the idea that software engineering should be a regulated profession. We discuss alternative forms of organisation that have proved popular among software engineers; particularly the self-organisation favoured by the Agile Software Development and Software Craftsmanship movements.

We explore the ethical considerations relevant to software engineering, including examples of problems in the field that have been described in the literature or the press as ethical failures. We compare these ideas of software engineering ethics with the codes of practice software engineers are expected to abide by in both research and academia, and find that the ethical properties relevant to the cases explored are mostly not covered in these codes. This motivates the question of how software engineers uncover, discuss, and resolve ethical issue in their work.

This chapter does not contain a complete literature review of the field of RSE, we review the literature and identify a research gap in chapter 3.

2.2 Research Context

RSE was originally proposed as a concept at the 2012 Software Sustainability Institute (SSI) Collaborations Workshop, in a breakout session on the "career track for software developers" chaired by James Hetherington [1]. The list of participants is not published, however the context is a national workshop of people in research with a focus on supporting software development. Observing the lack of recognition of software contributions to research, participants proposed "Research Software Engineer" as an identity for those creating software in research [2]. Research

Software Engineers could organise under a common banner to raise recognition of the important role they play in research, to define career pathways, and to identify and share best practice in their work. This recognition was stated to include certification and "an institution or professional Body", with the analogy of the British Computer Society's "Chartered eng." designation being suggested.

2.2.1 Software in Research

Contemporary researchers use many, wildly different software tools, from one-off scripts designed to support an individual experiment or analysis to email applications, word processors, and web browsers. Some of this software they create themselves (or work with RSEs and others to create it), and some represent "off-the-shelf" tools they acquire from external providers. Some of the software contributes directly to the processes and workflows of producing their research, while some is used for ancillary tasks including communication and calendar management.

As such, software can be categorised into "research software" and "other software"—even where the other software is used by researchers in research contexts—and the category of research software further divided into a typology. Multiple approaches have been proposed for this categorisation. In the 2014 UK Research Software Survey, Hettrick et al. define research software as:

> Software that is used to generate, process or analyse results that you intend to appear in a publication (either in a journal, conference paper, monograph, book or thesis). Research software can be anything from a few lines of code written by yourself, to a professionally developed software package. Software that does not generate, process or analyse results - such as word processing software, or the use of a web search - does not count as 'research software' for the purposes of this survey.[3]

Gruenpeter et al. identify the need to provide a bounded definition of research software, for the FAIR 4 Research Software initiative to agree how to apply FAIR principles[1] to the artefacts they identify [4]. They examine an inclusive definition of

[1] We define FAIR principles in section 2.4.6.

research software ("All code and software artifacts that are used, produced, or might be related to the research process in one or more stages of the research lifecycle and regardless of the layer of the software stack.") and an exclusive definition ("Software that was developed with the intention of being part of research."), and explore their application to particular case studies, to arrive at a flexible definition that encodes the FAIR principles of findability, accessibility, and identifiability:

> Research Software includes source code files, algorithms, scripts, computational workflows and executables that were created during the research process or for a research purpose. Software components (e.g., operating systems, libraries, dependencies, packages, scripts, etc.) that are used for research but were not created during or with a clear research intent should be considered software in research and not Research Software. This differentiation may vary between disciplines. The minimal requirement for achieving computational reproducibility is that all the computational components (Research Software, software used in research, documentation and hardware) used during the research are identified, described, and made accessible to the extent that is possible. [4]

In constructing this definition, Gruenpeter et al. rely on the layers of a "scientific software stack", as defined by Hinsen [5]. Hinsen describes four layers of software, increasing in specificity:

1. *Non-scientific infrastructure.* Tools including compilers, data-management libraries, and web browsers, that computational scientists depend on, but that aren't scientific in nature.

2. *Scientific infrastructure.* Domain-agnostic scientific computing packages, including mathematical and visualisation libraries.

3. *Domain-specific research software.* Tools that are used by research communities in particular fields to support methods, or implement models, used in those fields.

4. *Project-specific software.* Scripts, workflows, special-purpose libraries, and other software that scientists write for their particular research projects.

Software in layer 1 depends in turn on an operating system (for example, GNU/Linux or Microsoft Windows), which itself depends on the hardware it runs on. Hinsen's motivation in constructing the "stack" metaphor in [5] is to argue that the ongoing utility of software at any level of the stack depends on the stability and compatibility of all the layers below it in the stack, and to introduce the idea of "software collapse" when a low-level change causes problems at higher levels. He describes risk evaluation and management strategies for scientific software.

Nieuwpoort and Katz take a different approach and categorise research software based on its role in the research process [6]. They identify the following positions for research software to occupy:

- A component in (physical or virtual) instruments for data acquisition and processing.
- An instrument itself, embodying a computational method, model, simulation, or experimental platform.
- A data-analysis tool, supporting model fitting, search, anomaly detection, and other modes of data analysis.
- Presenting research results, for example generating plots, or visualising data statically or interactively.
- Supporting workflows through integration and automation.
- Infrastructure that enables other categories of research software, whether that infrastructure is created specifically for research or as a general-purpose utility.
- Facilitating research-oriented collaboration on software, papers, data, computing, citizen science, or in other areas.

Yehudi et al. similarly identify categories of research software based on the software's role in research, with three high-level areas—modelling, simulation, and data analytics; technology research software; and research infrastructure software—organised into sub-roles [7]. Unlike [6], they explicitly bound their infrastructure category to exclude general-purpose utilities. They further categorise research software by its stage, developer community, method of dissemination, criticality, and maturity. Such categorisation supports evaluating research software based on aspects of its development project rather than its purpose, as in the OSS Watch Software Sustainability Maturity Model [8].

2.2.2 Artificial Intelligence in Research Software

Generative AI tools—deep machine learning models trained on large amounts of data, that generate statistically probable outputs in response to prompts—have the capacity to change how knowledge workers diagnose a problem, infer potential solutions, and take action to treat the problem [9]. Authors have explored the transformative effect such tools can have on software creation [10] [11], and this can include the creation of research software.

Generative AI, including large language models (LLMs), have been applied in multiple research domains, including plasma physics [12], biodiversity [13], and earth sciences [14].

2.2.3 Stakeholders for Research Software

As contributions to research projects, research software involves a multitude of stakeholders. We follow the example of the Science and Technology Studies (STS) field, as described in [15], in including people and organisations who 'translate' the research into use and who use the research knowledge.

- RSEs themselves. We define RSEs as people who use the phrase "research software engineer" about themselves (whether formally, for example, as a job title; or informally, for example, when people discuss their work with others.

- "Proto-RSEs". There are people whose work responsibilities overlap significantly, or even completely, with those who call themselves RSE, but who consider themselves researchers who work on software, computational scientist, or otherwise don't describe themselves as research software engineers. One RSE blogger referred to such people as "proto-RSEs" Blog 2.

- Other researchers. We include people who work as research assistants in research groups that use or create research software; doctoral research students in such groups; and principal investigators and co-investigators who run projects to produce research software, projects that create research software as a secondary output, or projects that depend on research software.

- Software engineering researchers. People who seek to understand and improve research software [7].

- Students. In addition to producing software in or for research groups, students may need to acquire software engineering skills, or achieve course credit in software topics, as part of their studies.

- Research-producing organisations (RPOs). These are organisations that have research goals and hire researchers (and, possibly, RSEs) to further those goals, including universities, private companies, and government research institutions. As such they may contain people who form professional relationships including power relationships with RSEs, for example, contractual, line management, career development, and performance-reviewing relationships.

- Journal publishers. A research journal may publish research that uses software, or papers that represent or describe research software as their primary contribution. In such situations, RSEs can be authors or (credited or uncredited) contributors to the published research, and they or other people may act as peer reviewers and editors.

- Software producers. Research software can incorporate or build upon software from other sources, including both open source and proprietary software. As discussed in section 2.2.1, research software can include libraries and tools that other research software projects rely on, so RSEs can themselves be in producer-consumer relationships with other RSEs and researchers.

- Funding bodies. Research software is often created as either the primary or an ancillary output of a research project. As such, the organisations who fund the projects—and who review project proposals to decide what projects to fund and to what levels—are stakeholders in the research software. In the UK, many research projects are funded by government-run research councils that comprise UK Research and Innovation (UKRI), or by other agencies of the government's Department for Science, Innovation & Technology. Other funders include private companies, research charities, and organisations from outside the UK; for example, projects that are part of Horizon Europe receive funding from the European Union.

- Support organisations. Institutions such as the Software Sustainability Institute and Society for RSE advocate for and assist in the production of research software, and careers for RSEs. We discuss these bodies in more detail in section 2.4.1.

- Research users. Due to the many varieties of research undertaken, there are a broad range of people who could either use research software or—more frequently—its results, outside of a research context. These can include: policy makers who rely on research outputs to make judgements about their fields; medical practitioners and patients who administer and receive interventions guided by research; and commercial organisations who design innovative products based on "translating" research results.

2.3 The Development of Software Engineering

While the concept and role of RSE is a recent development in computational research, its reliance on the idea of "software engineering" connects it to a complex history in which software creators, their managers, and their customers have struggled to define the roles and responsibilities of those who make software, and their position in society.

2.3.1 The Introduction of "Software Engineering" and the Software Crisis

The phrase "software engineering" was first applied to the production of software as a 'provocative' choice by the NATO Science Committee Study Group on Computer Science, in recommending an international working conference on software manufacture. The implication of the term was "the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering" [16].

The name stuck, and by the end of the 1960s people were being hired as "software engineers", rather than "computer programmers", according to software methodology practitioner Capers Jones [17]. Software engineering became the name of a field focusing on the economical production of quality software. Reflecting the difficulty of achieving this goal, the idea of a "software crisis" pervaded the literature from approximately this time, into the subsequent millennium.

Software's ongoing crisis is characterised by cancelled projects, cost overruns, and software delivered that is so defective as to be useless in the field. While a software engineering researcher has called the source data used to inform and perpetuate the crisis narrative into question [18], high-profile software project failures still reinforce the perception that software engineering is an immature discipline unable to reliably produce quality output. One industry organization estimated the cost to the U.S. economy of poor-quality software in 2020 to be approximately two trillion U.S. dollars [19].

Proposals to address the crisis abound. One industry business leader suggests a stronger theoretical underpinning is necessary, with software engineering being supported by a "software science" based on formal methods and proofs of correctness [20]. Another recommends improvements to standardisation of interfaces and a component model of software production, akin to the move from cottage manufacture to large-scale factories in the industrial revolution [21]. The Software Engineering Institute at Carnegie-Mellon University proposes a maturity model, in which software engineering organisations define, control, and optimise their processes [22].

While some authors see software as being in "chronic crisis" [23] [24], others question the existence of the crisis at all and point to the broad availability of working, profitable software as counter-claims. Bertrand Meyer satirises the Agile software consultant—selling their preferred methodology as a solution to a software development manager's ills—waking up to an alarm set on their smartphone, programming their microwave oven to cook their breakfast, relying on various other computerised systems and software packages to navigate both work and personal life, before opening a document in a cloud-based collaborative editing application to write that "You have been ill-served by the software industry for 40 years—not purposefully, but inextricably". [25, p. 30]

In a conference paper exploring the origins of software engineering, Thomas Haigh finds that the idea of a sector-wide "software crisis" was retroactively constructed and propagated [26]. The only mention of a "software crisis" in [16, p. 120] is in editorialisation of a discussion in which "widely differing views" were presented, though some delegates talked about a "software gap" between the aspirations of computer users and the ability of professionals to deliver functioning systems. Edsger Dijkstra, in his 1972 acceptance speech for the ACM's Alan M. Turing award, presented the narrative of the Garmisch conference being where "open admission" of the software crisis—hitherto a "blasphemy", was first admitted [27].

Whether or not the software sector started or remained in crisis, software engineering as a field sought to provide a sound, economically-realistic basis to software development and to support the large-scale development of computer systems that

purchasers were coming to expect. RSE could be seen as a continuation of this development—the application to research of software engineering—or alternatively as a new movement to provide a sound basis to the application of computers in research: the engineering of research software.

In this latter conception, RSE can be seen to respond to perceived problems in research capability, including the reproducibility "crisis" reported in some fields [28].

2.3.2 Debate Over the Professionalisation of Software Engineering

Proposals to address the software crisis sometimes come with calls to professionalise software engineering [23]. If software engineers need to use new practices and skills to develop high-quality software—taking formal methods as proposed by Gibbs and by de Champeaux as an illustrative example—then software engineering should re-form as an exclusive or licensed profession, only open to those willing to adopt the new practices and adhere to higher standards.

Software Engineering developed some of the trappings of a distinct profession in the decades following the NATO conference. There are professional associations for computing professionals with dedicated sections or special interest groups for software engineers, including BCS: the Chartered Institute for IT in the UK, and the U.S.-based IEEE Computer Society and Association for Computing Machinery. The BCS permits members to register for recognition as Chartered IT Professionals and provides professional certifications in multiple computing-related disciplines, none of which is software engineering itself but many relate to areas of the Software Engineering Body of Knowledge (SWEBOK), including software development and information security [29].

There are accredited degrees in software engineering delivered at many institutions, including the University of Oxford [30] and University of Swansea [31]. However, in very few regions in the world is software engineering a licensed profession, bringing exclusivity of practice from the state and esteem from the public [32].

Indeed the debate over whether software engineering should be a licensed profession is not settled, and continues to play out in the literature.

The SWEBOK is published by the IEEE, and its authors describe it as "an essential step" towards describing and accrediting a curriculum which can be used for licensing examinations [33]; i.e. for introducing an exclusive profession in which only licensed practioners can participate. An ACM task force on software engineering licensing recommended that the ACM withdraw from work on the SWEBOK as it did not include safety-critical software in its scope, which the task force identified as the context most in need of best practice guidance [34].

Frailey pointed to two benefits of licensing software engineers: it would force engineers to define, codify, and advance the profession, and it would stop unqualified or incompetent engineers from harming society [35]. Knight and Leveson [36] argue that the idea of licensed professional software engineering should only apply to "safety-critical" systems as does Kruchten [37], but neither define the boundary between such systems and other software.

Knight and Leveson further argue that, given the word "engineering" in its title, software engineering would be entered by passing an accredited engineering exam but that the engineering body of knowledge is too broad and irrelevant to a software engineer to be a reasonable basis for accreditation. Laplante observes that in some regions where software engineering is already an exclusive profession, waivers are given to the broader engineering requirements for individuals with software engineering-relevant educational or experiential credentials [38]. He suggests that practitioners in professional societies should get involved with the licensing discussion regardless of their position on the debate, lest governments impose policy changes without consulting experts in the field.

One result of the non-exclusive state of professional software engineering is that while the professional societies all publish codes of professional ethics, an individual need not subscribe to, uphold, or even be aware of any of those codes to practice software engineering or to hold a job with the title "software engineer".

Movements have arisen that loosely organise software engineers around particular principals or practices, but that tacitly or explicitly reject the professionalisation of the field. McCalla argues, based on the experience of licensing software engineering in Canada, for the simple opposition to licensing. His alternative is that the professionalism of any individual engineer is merited, assessed, and prosecuted based on their own experiences, professional contacts, references, performance, and demeanour [39].

More recently, influential members of the ACM have been advocating for a more active role in professional regulation, suggesting that the dominant narrative that software engineering would not benefit from professionalisation is losing some legitimacy. The former president, Vinton G. Cerf, proposed in an editorial column in December 2023's *Communications of the ACM* that "ACM could be an ideal source of experts who could be called upon" to give expert testimony in court cases where computing is an important factor [40]. In the next issue, former *Communications* editor-in-chief Moshe Y. Vardi published a column with the provocative title "Computing, you have blood on your hands!" He argued that lax moderation policies on social media and the dominance of computing by a small number of large corporate interests led to rising hate speech, support for human rights violations in Myanmar, and a teenage mental health crisis, concluding "it is time for all computing professionals to accept responsibility for computing's current state" [41].

To understand whether formal professionalisation and certification of RSEs—an approach suggested in the breakout group discussion where the term originated [1]—would promote and disseminate desirable professional practice, we should first understand what practices RSEs and the stakeholders identified in section 2.2.3 consider desirable. We also need to investigate how those involved in a research software project organise their work, teams, and development, to understand the similarities and gaps between the existing approach, "professional" software engineering, and the alternatives promoted by industry practitioners and discussed in sections 2.3.3 and 2.3.4 below.

Software engineering became associated with a Taylorian approach to management oversight, and processes that trace every activity from requirements statement to deployment, in an attempt to provide a repeatable, reliable basis for creating software. Subsequent movements rejected this process-centric approach, espousing flexibility and shifting responsibility and authority from the management to the team of implementors. In the next two subsections, we note the rise of two of these movements and the centrality of the "manifestos" that leaders published to attract people to their causes: the 2001 Manifesto for Agile Software Development, and 2009's Manifesto for Software Craftsmanship.

2.3.3 Agile Software Development

Commercial software engineering changed significantly following the publication of the Manifesto for Agile Software Development in 2001 [42]. The authors of this manifesto—the Agile Alliance—recorded that through developing software "and helping others to do it", they had come to value certain properties of a software project over others—for example, valuing responding to change over following a plan. They proposed a collection of twelve principles which collectively form a disciplined approach to software development that conforms with the values they enumerate.

Certain practices in software engineering have come to be known as "agile practices". There is no explicating theory for these practices to arise in agile software development, although they are promoted within methodologies that support the values and principles in the agile manifesto, or observed among teams self-describing as "agile", and were not commonly practiced by engineering teams following traditional sequential methodologies. Extreme Programming (XP) lists 29 "rules" that govern practice on teams following the methodology [43].

Jacobson et al. argue that practitioners' theoretical grounding for the methods used in software engineering are too shaky to accept that a methodology or paradigm can be accepted wholesale [44, pp. 26-28]. They argue, from their perspective as software engineering methodological consultants and academics, for rejecting methodological thought and bundles of associated practices. Software engineers and managers should instead take a systems thinking approach and evaluate

the weaknesses of a software engineering process against its desired state before introducing and evaluating particular practice interventions [op. cit., pp. 28-29].

In his criticism of agile software development, Bertrand Meyer reviews some of the key texts promoting the methodology and finds that many present over-generalised claims backed only by anecdote and allegory [25]. He argues that the reason for this is the limited availability of evidence-based results in software engineering; many of the empirical results that are available drew participants from undergraduate software courses and cannot be expected to have ecological validity in a professional setting.

Meyer doesn't provide an argument for why this should be the case: it may be that a lack of empirical evidence for software engineering practices follows from a lack of demand for empirical validation, or that a low reliance on evidence-based argumentation is caused by a poor supply of evidence (or that these two relationships generate a vicious cycle).

2.3.4 Software Engineering as a Craft

A popular alternative among practitioners to the organisation of software engineers as a licensed profession is organisation as a craft, with master software craftsmen teaching their skills to apprentices who then become journeymen working on their own projects. In this model, software construction is a skilled craft comprised of tacit knowledge that must be gained through experience, and not an engineering discipline with a scientific-theoretical underpinning that can be documented, examined, and certified [45]. The craftsmanship model of software development centralises the practitioner who has gained arcane, tacit knowledge that apprentices acquire through observation and collaboration, and rejects the idea of software engineering as a controlled, replicable process that can succeed through judicious management.

A manifesto for software craftsmanship [46], modeled explicitly on the manifesto for agile software development, describes the movement's imperative as "raising the bar of professional software development" by focusing on factors including "a

community of professionals" and "well-crafted software". At time of this writing, there were over 31,000 signatories to the manifesto [47].

The guild metaphor described in [45] is compatible with the reality that many professional software engineers already treat the discipline as a craft. The authors of the manifesto for agile software development defended the document's authority through their own reputations: "We are uncovering better ways of developing software by doing it and helping others do it." [42] There is no evidence-based argumentation or theoretical explication that leads to the values and principles outlined: they are offered based on the success of the authors.

Where an author has sufficient standing in the professional community, practices that they propose may become adopted without explicit recourse to sharing their credentials. This is consistent with the perception of these authorities as "master craftsmen". Kent Beck, originator of Extreme Programming and one of the co-authors and signatories of the Agile manifesto, introduced a new software testing practice called "test && commit || revert". His justification for engineers to try this practice is that "It's cheap [and] you're bound to learn something" [48]. On August 30 2021, a search for "test commit revert" on the blog search engine http://www.blogsearchengine.org/ yielded 25 results for articles, workshops and podcast episodes by other authors discussing or promoting the technique.

When considering RSE as an activity, role, or profession, it is important to discover what research software engineers and other stakeholders perceive by the "software engineering" in RSE.

2.3.5 The Ethics of Software Engineering

An important aspect of the professional organisation of a field is how (and indeed whether) practitioners perceive and navigate ethical issues related to their role's unique position in society, and whether ethical norms are propagated and enforced. Following Aydemir and Dalpiaz [49], we define the "ethical-ness" of a software engineering initiative to refer to its conformance with the ethical values of its many stakeholders. Aydemir and Dalpiaz identify users, software engineering

professionals, and software development organisations, as different actors all seeking "ethical harmony" which may bring consistent or conflicting ethical requirements to the initiative. These requirements may represent desiderata in the software artefacts produced, or in the engineering processes enacted in the production.

The problems that arise in software engineering—and thus in RSE—which can be seen to have an ethical dimension take many forms. Here we organise a "taxonomy" of ethical issues in software engineering into four distinct categories.

The Software Engineer's Interactions as a Professional in their Workplace

Software engineers may be employed in, or contracted to, a workplace in which they are expected to engage in civil and professional discourse with their colleagues, and avoid bullying, discrimination, and harassing behaviour. In addition they may be expected to protect their employers' copyrights, trade secrets and other intellectual property, and avoid illegal corporate practice including anticompetitive behaviour, insider trading, and theft of third party intellectual property.

In 1997, former Octel software engineer Shahryar Soroosh was found guilty of insider knowledge, using internal information about delays to an Octel software project to gain more than $500,000 through short selling the company's stock [50]. He was ordered to repay the profits plus interest to stockholders.

In 2017, Google employee and software engineer James Damore was fired for publishing an internal memo in which he asserted that women were intrinsically less suited to software engineering than men [51]. Google terminated Damore's employment, though this action may itself have contravened employment law in California, which prevents firing employees for their adoption of political courses of action.

In 2020, software engineer Simon Finch was tried for breaking the Official Secrets Act by leaking details of a missile system he had worked on as a contractor [52]. Finch was found guilty, and sentenced to four and a half years in prison [53].

In a recent survey of institutional codes of conduct, the author of this thesis found that the majority of ethical imperatives contained in the codes of U.S.-based publicly traded technology companies fit into this category [54]. The codes of

practice are written to generally cover employees and contractors at the corporations, and there is little connection in these codes with the software produced by those corporations (more accurately by the people working for them). The majority of imperatives relate to conformance with securities trading and market fairness regulations: for example resolving commercial conflicts of interest; not bribing government agents; and not engaging in insider trading.

This author compared this situation against codes of conduct in U.K.-based universities, which mostly fell into the "public good" category discussed in subsection 2.3.5.

Using the Software Engineer's Skills for the Public Good

The first section in the joint IEEE-ACM code of ethics is about the software engineer's responsibility to the public [55]. In addition to moderating personal and employer interest with the public good, the code asks software engineers to accept full responsibility for their work; disclose failures of software practices to uphold the public good; cooperate in attempts to address problems that arise; consider diminished access to the software's benefit caused by disability, lack of resources, and other factors; and to consider volunteering their professional skills to good causes.

The infamous case of the Therac-25 medical accelerator, which caused multiple fatalities and serious injuries due to software defects controlling the safety features [56], represents multiple failures to act in the interest of the public good (leaving aside any discussion of whether the software had been designed and implemented to a professional standard). As Leveson and Turner found in their post-mortem report, the original engineer did not cooperate in external inquiries into the problems, and had not provided sufficient documentation to enable quality assurance of the software created. The company, AECL, did not release the software for external scrutiny after serious accidents had been reported, citing their proprietary rights. A safety report produced by AECL explicitly excluded software failure modes from consideration.

In 2015, software engineers working for Volkswagen were found to have crafted a software-based "defeat device" that allowed diesel engines to produce many

times the legally-permitted levels of Nitrous Oxide pollutants in normal operation, potentially increasing mortality due to air pollution in regions where vehicles using these engines were sold [57]. Engineers and managers on the responsible team were fired, and eventually Volkswagen's CEO Martin Winterkorn resigned, though he claimed no knowledge of the defeat device [58].

In the survey of institutional codes of conduct previously mentioned, the author of this thesis found that U.K.-based universities typically have codes of "research integrity" modelled on the Concordat for Research Integrity [59]. These codes promote some aspects of the responsibilities researchers have to their broader communities, including ensuring the accuracy of the scientific record, and the humane treatment of human participants and animals in their research [54].

As with the commercial codes discussed in subsection 2.3.5, no connection is made between the ethical imperatives of university codes of research integrity and the software created in those universities, despite the increasing dependence of research on software. The question of how issues raised in relation to research integrity—or other ethical dimensions in research software engineering—are surfaced and resolved within research software projects remains unexplored.

Creating Software Systems that Support the Ethical Values of their Users

The software engineering team working on a software system are not the only people to interact with that system. The users of the system may also make decisions supported by, or influenced by the presence of, the software, with potential ethical impact on themselves, other users, the developers, or unconnected third parties.

In relation to the automation of power plants, Kawai observes that the ethical responsibilities of the engineers extend to the skill levels and job satisfaction of the operators whose work is affected by the automation, alongside the safety and correct operation of the automatic system [60]. In the medical sphere, software-based "virtual patient simulations" can be used to present ethical case studies to medical students [61].

Introducing software into a system can change the nature of ethical decisions or behaviours made in that system. Automated plagiarism detection software reduces the effort teachers and editors put into discovering genuine cases of plagiarism, though the software is known to produce false positives; students can also use the software to discover whether their work will be detected as plagiarism, resulting in an arms race in plagiarism capabilities [62].

Many research software projects have a user community beyond their research group. As one example, the Global.health project[2] provides a curated "line list" of data about Covid-19 infections that is used by researchers, journalists, and policy makers [63]. The project must take the needs and concerns of these groups into account, as well as those of the data providers and the individuals whose cases are recorded. There is value in exploring how these concerns are captured and addressed across the RSE field, and the impact they have on the software created.

Ethical Issues Arising from the use of Artificial Intelligence and Machine Learning Systems

Artificial Intelligence and machine learning systems represent a special case of software systems that can introduce ethical problems, particularly where the machine learning software supports or replaces human decision making. As noted in section 2.2.2, such systems are increasingly finding application in research software contexts. One introductory textbook posits a short list of "fresh problems" facing AI implementors:

- unemployment caused by automation;
- social problems associated with changes in work schedules;
- loss of humanity's sense of uniqueness;
- use of AI toward undesirable ends;
- unaccountability in application of AI; and

[2] At the time of writing, the author of this thesis was the software lead on Global.health.

- "the rise of the machines" and the end of humanity [64].

Biases in the training data can lead to biased decisions, as in the case of the face recognition software at Google that detected "gorillas" in photos of black people; a problem Google "fixed" by removing gorillas and other primate species from the list of known categories [65].

A similar situation arose when Amazon trained a machine learning system to filter job applicants' resumes, with training data from Amazon's hiring history. The trained system exhibited a clear bias for male candidates, rejecting graduates from all-female colleges and resumes that mentioned a candidate's female gender [66]. It should be noted that the same technique can be used to highlight existing biases with a view to correcting them: a machine learning system trained on astronomy papers written by male first authors predicted that papers written by female first authors would attract 4% more citations, when the measured citation rate was 6% lower [67].

Once an artificially intelligent software agent takes over a decision from a human operative, it is not clear where the responsibility lies for any ethical lapses—perceived or actual—in the decisions made. In 2016, the United States National Highway Traffic Safety Administration (NHTSA) investigated a fatal crash that occurred in 2015 involving a Tesla car operating in the self-driving 'Autopilot'[3] mode [68]. While the NHTSA did not find any defect in the operation of the Autopilot feature, they did note that many drivers will not read any manufacturer-supplied information about the capabilities and expectations of autonomous driving modes and say that "Manufacturers therefore have a responsibility to design with the inattentive driver in mind." [69] They do not indicate whether the accident under investigation was due to a human-machine interface design that failed to take the "inattentive driver" into account, nor whether Tesla, its management, or its engineers would have been held to account had this been the case.

The capability and applicability of machine learning approaches to many problem domains expanded rapidly with the invention of LLMs, an example of generative AI

[3]Autopilot is a Tesla marketing term. The feature supports Level 2 driver assistance, in which full driver alertness is still required.

supported by a flexible *foundation model* that can be applied to many tasks. LLMs were enabled by the Transformer architecture, which increased the parallelisation of the model and reduced training costs while improving the quality of the output [70]. Language models are trained on large corpuses of text, and fine-tuned through prompts and instructions to interpret input text and generate text responses [71].

One risk associated with LLM application is *prompt injection*, in which someone overrides the developer's instructions and convinces the model to perform a different task, or to give results that are inconsistent with the user's goals [72]. Such prompt injection can be indirect, in which malicious data is introduced in the training material rather than alongside the user's input [73].

Another problem is *hallucination*, in which the model produces output that appears factual but that was entirely generated by the model and is ungrounded [74]. Multiple situations are reported in the legal sector, in which lawyers submitted motions to courts that were wholly or in part generated by LLMs, and which contained hallucinated references to fictitious cases [75].

Additionally, the copyright issues involved in using creative works as training material for LLMs are not yet resolved, and raise ethical questions about whether creators should have a say—or receive compensation—when their works are used in training models that can then generate similar works in their output [76] [77].

Machine Learning and other techniques from artificial intelligence enter into research software in two different ways: researching the architecture, algorithms, and techniques of artificial intelligence; and applying such techniques as software tools in other areas of research. How researchers and RSEs on projects using artificial intelligence address the "fresh problems" listed in [64], and address the risks to good scientific practice discussed in [78], will become increasingly important as the tools enter the mainstream and become easier to deploy.

2.4 Research Software Engineering

With its genesis at a workshop held in a university for members of the academically-focused SSI, many Research Software Engineers (RSEs) are employed in academic

institutions and work with or as academic researchers. Academia has its own professional standards (often with different interpretations and emphases in the various research disciplines), career pathways, and governance structure. The Software Sustainability Institute's deputy director argues that the academic system does not adequately recognise the contributions made by software engineers—along with other contributors—to academic outputs [79].

In using the phrase "software engineering" in the title of RSE, leaders and practitioners connect their professional identity with the crises, debates, and problems observed in software engineering and summarised in the above sections.

2.4.1 Organisational Support

As noted above, the term RSE was originally introduced in 2012 at an event supported by the Software Sustainability Institute. The SSI advocates for recognition of the contribution research software and its creators make to research, improving software engineering practices in research, and providing software training to all researchers [80]. The institute reports that its most recent round of funding raised £6.6M in 2018-2023, plus other income raised by collaborating directly on funded research projects [81]. The total budget for UK Research and Innovation (UKRI)—the government's research investment department—in the financial years 2018-2019 to 2022-2023 was £37.68B [82]; so software sustainability represented approximately 0.02% of UK government research spending.

In 2019, the Society of Research Software Engineering was founded as a charity that succeeded the existing UK RSE Association, which was created in 2013. The society is a grass roots organisation funded by member subscription that has similar aims to the SSI, "to increase software skills across everyone in research, to promote collaboration between researchers and software experts, and to support the creation of an academic career path for Research Software Engineers" [83].

Both the SSI and the Society of RSE are based in the UK and focus on research software activities undertaken in the UK. We acknowledge that similar organisations

exist in other countries, but as we focus in this thesis on research software in the UK we do not go into detail on the research software ecosystem elsewhere here.

2.4.2 Policy Issues

Research software itself can be at the centre of deep policy controversies, meaning that the scrutiny applied when reviewing software can be a matter of public interest, beyond the professional demands of peer review associated with publishing papers that rely on the software. A recent high-profile example is the epidemiological modelling software created by Ferguson et al. that informed the UK government's decision to impose "lockdown" stay-at-home orders in 2020, in response to the COVID-19 pandemic [84]. Jalali et al. assessed the transparency of 29 COVID-19 models including [84], and found that no model satisfied all of their criteria, with the Ferguson model failing 14 of 27 tests including provision of data, model equations, codes, documentation, identifiability of the software used, and declaration of conflicts of interest [85].

The funding UK higher education institutions receive from the government depends in part on the outcome of their assessment in the periodic Research Evaluation Framework (REF). The next REF assessment occurs in 2028, and the initial decisions on the framework include "an expansion of the definition of research excellence to ensure appropriate recognition is given to the people, culture and environments that underpin a vibrant and sustainable UK research system". [86] This shift in emphasis to include recognition of more diverse contributions offers an opportunity for RSE leaders to demonstrate the benefits that RSEs bring to research.

2.4.3 RSE Skills Acquisition

RSEs need to acquire software engineering knowledge and capability and stay up to date with developments in the field, and potentially train researchers and junior colleagues in software engineering practice, alongside other skills that their roles require.

A 2020 workshop held online for RSE community members identified that understanding the skills needed for early-career RSEs, and whether RSE training

should be part of every student's curriculum, were high-priority questions for the workshop participants [87]. In 2023, Goth et al. identified three broad categories of "foundational RSE competencies": software and technical skills; research skills; and communication skills [88].

They list multiple organisations and projects that develop educational materials in research software, including the Carpentries, UNIVERSE-HPC,[4] and CodeRefinery.

Grossman et al. outline the scope of a "software management plan" (SMP), comparing SMP templates from various sources to identify clusters of topics covered by the templates that represent items a research software team should include in their management plans [89]. The list of topics—and, by inference, the skills that a successful research software development team should include to competently address such topics—is broader than Goth et al.'s list, including administration, documentation and versioning, legal and ethical aspects, performance and security, preservation and sharing, related objects (other artefacts including the plan itself), software description, and technical infrastructure.

In each case the list of skills is broader than those expected of a general software engineer, with Goth et al. listing the IEEE Software Engineering Body of Knowledge as a single item, "classical software engineering skills", in their category of software and technical skills.

Various community hubs and centres of practice also act as forums to share and develop knowledge among research software professionals, in the UK these include the Science and Technology Facilities Council's Computational Science Centre for Research Communities [90], which supports shared software infrastructure for multiple research domains through Collaborative Computing Projects.

RSEs share experience, and teach skills, among each other through community-based activities included the Society of Research Software Engineering conference [91] and the Code for Thought podcast [92]. Universities and other RPOs offer training and career development for their employees, including RSEs; for example,

[4] I was a member of the UNIVERSE-HPC advisory board.

Oxford University's RSE group offers training to researchers—including RSEs—who are members of the university [93].

2.4.4 Research Software Sustainability

As seen in sections 2.2.1 and 2.2.3, research software can be created at a confluence of short-lived circumstances: created by students or research assistants whose association with a particular project is of limited duration; and created to solve the immediate needs or a project or even a single publication within that project. Such circumstances—along with inconsistent sharing of software artefacts associated with research publications—can lead to problems of *sustainability*, in which further development of the software or replication of its results is difficult, expensive, or even requires re-creating the software from scratch which risks behavioural divergences from the original software.

Katz et al. discuss the importance of sustainability in research software, and recognise that supporting people in ongoing involvement in the software's maintenance is key to making software sustainable [94]. Ross Gardler at OSSWatch proposes a Software Sustainability Maturity Model (SSMM) [8]. The SSMM defines nine levels of maturity from "Seed" to "Dispersal" (based on the life cycle of a pear tree), and compares the expectations at each level with those of other models; the NASA Earth Science Data Systems' Reuse Readiness Rating, and the Software Engineering Institute's Capability Maturity Model (CMM).

The Amsterdam Declaration on Funding Research Software Sustainability (ADORE) states the importance of the role of funders in sustaining research software, and in cultivating a research culture that supports sustainability [95]. At the time of writing, the declaration had 55 institutional and individual signatories [96].

2.4.5 Responsible Research and Innovation

The codes of ethics and other forms of ethical consideration described in 2.3.5 are typically designed to apply to the profession of software engineering generally. The same professional code for life-critical medical software engineering applies

to engineers working on autonomous weapons systems, testers of fluid dynamics codes and managers of teams developing freeplay mobile gaming apps. Such codes only consider a narrow interaction between software engineers, their employers, and society at large; they do not give engineers a way to consider the various stakeholders in a project and how their needs overlap or conflict.

Research software engineers and their collaborators have particular considerations to account for in their work, motivated by the research context. One way of thinking about these considerations and implementing supportive actions is the Responsible Research and Innovation (RRI, or Responsible Innovation) approach. RRI covers broader themes than research ethics, including: the motivations for undertaking research; impact and consequences on society and stakeholders; and the appropriate communication of interests and concerns between the practitioners and other stakeholders [97].

Researchers have already started applying RRI thinking to ICT projects. Stahl et al. argue that RRI should supplant computer ethics as the "reference discourse" for ethics in information systems work, to improve awareness and capability among practitioners [98]. No work has yet been done to understand the reference discourse of ethics among RSEs or their current level of awareness. A study from a constructivist perspective on how research software engineers perceive and navigate issues of ethics and responsibility would illuminate the opportunities, appropriateness, and potential benefits of a paradigm such as RRI to the field.

2.4.6 FAIR for Research Software

Concurrent to the development of RSE is the creation, dissemination, and adoption of FAIR guiding principles in research data management. This represents an analogous situation to research software engineering: practitioners and leaders observed that an important contribution to modern research (software engineering in the case of RSE, data stewardship in the case of FAIR) was unrecognised in the context of rewarding academic contributions and advancing career structures. There are

opportunities for the two movements to learn from each other and to collaborate as data management and software are closely related activities.

In the context of research data management, the FAIR Guiding Principles—that data must be Findable, Accessible, Interoperable, and Reusable, were introduced in 2016 by stakeholders in the FORCE11 community (The Future of Research Communications and e-Scholarship) to guide the observed rise in disparate data management requirements toward common goals, and to place emphasis on automated support for data management practices and away from human effort [99]. The guiding principles were introduced by a collaboration of 53 authors from an internationally-distributed background of universities, research institutes, academic publishers, and independent advocacy organisations.

As suggested by the title, the FAIR Guiding Principles are intended to be normative, suggesting practices that the authors believe to be beneficial without enforcing those practices. The FAIRsharing team led by Oxford University's Data Readiness Group describes FAIR as "a community approach" to developing data management standards [100].[5]

Currently, adoption of FAIR Guiding Principles is discipline-dependent and in many situations voluntary. For example, the Nature Portfolio of journals requires that authors "make materials, data, code, and associated protocols promptly available to readers without undue qualifications", but only for certain classes of data do they require that authors deposit the data in well-known data repositories that generate Digital Object Identifiers (DOIs) the authors and others can cite when they refer to the data [101].

Multiple authors apply FAIR principles to research software, in an extension to its context in research data. Hasselbring et al. distinguish between computer science, and computational science, in discussing community norms for software availability and reuse [102].

Katz et al. identify multiple working groups addressing "FAIR for research software", and gaps in its application including a lack of a metadata authority

[5] One of my supervisors, Prof. Susanna-Assunta Sansone, leads this group and is the lead author on the cited article.

and no community consensus over software identification or metadata requirements [103]. More recently, members of the EOSC association have produced guidance for metadata standards for research software [104].

2.5 Conclusion

The field of RSE connects research software with the field of software engineering, which while long-established with multiple professional associations, contains active debates and contradictions relating to professionalisation and exclusivity, ethics, and professional conduct. RSE additionally encounters active discussions in the performance of academic work, including how researchers interact ethically with participants and other stakeholders, and how digital artefacts including software products are made available and usable by the research community.

With this context in place, we turn to the literature in chapter 3 to understand the state of the art in research on RSE, and to identify research gaps that we can explore and contribute to.

3

Literature Review

Contents

3.1 Introduction

We conduct a review into the literature that describes existing study of Research Software Engineering (RSE) as a profession. This review uses as its scope the context and background presented in chapter 2, which explored the development of Research Software Engineering and the role of Research Software Engineer.

We review such research as has been published on Research Software Engineering (typically in the closely-related field of "scientific computing") to understand what is currently known about how research software is made, and to identify a gap suitable for our investigation. We connect this work to the themes and theories found in the more general research on professions and on professional software engineers.

We then look to the literature on two community-driven changes in research culture—the principles of Findability, Accessibility, Interoperability, and Reproducibility (collectively FAIR) data management, and Responsible Technology and Responsible Research & Innovation (RRI)—as they provide interesting analogies to the development of RSE.

Subsequently, we review research on software engineering to understand how professional practices are evaluated and propagated among software engineers. This review focuses on software engineering as professional practice, not the basic research in the software engineering research domain, for example, support tools such as static analysers and programming languages. We organise the review according to the kind of data examined in the research (purely quantitative, purely qualitative, or mixed) as these approaches often arise from different epistemologies and thus explore the field in different ways. We examine the forms of evidence and argumentation used within the field, and the scholarly literature researching the ways in which software engineers value and practice their work. This understanding of how software engineering has been researched permits an argument by analogy in RSE.

We investigate the broader field of the sociology of professions to find research methods and results that could be translated to RSE. The literature we consider

comes from the functionalism school of thought in the sociology of professions, the professionalisation of occupations, and contemporary research on knowledge work. Recently, grounded theory methods have been applied, particularly in healthcare professions, by researchers to provide more general explaining theories of how social, technical, and political factors are perceived and navigated by practitioners. We also review alternative models of occupations and professions, including neo-institutionalist, Marxist, and neo-Weberian models, and the theory of communities of practice.

Software engineering saw professionalisation efforts in multiple regions, particularly the United States, United Kingdom, and Canada, through the 1980s and 1990s. Historians of software engineering have written academic accounts of these initiatives that connect the professionalisation of software engineering with the theories of fission, conflict, and collaboration present in the sociology literature.

Finally, we present conclusions on the gaps in our understanding of RSE, and the analogous work that has been done in other fields that can inform techniques for investigating the profession.

3.2 Research Software Engineering

In this section, we explore the literature on the importance of software (and particularly the construction of custom software) to modern research practice, and the ways in which RSEs connect research software with software engineering.

In 2009, Hannay et al. discovered that while 84.3% of scientists surveyed reported that developing scientific software was important or very important to their research, spending on average 30% of their time developing scientific software, only 34.4% said formal education, and 13.1% formal training, was important or very important to their software development education [105]. Similarly in the arts and humanities, the Arts and Humanities Research Council's digital/software requirements survey found that 83.1% of participants use research software, but 28.8% felt that they have received sufficient training to develop reliable software. The Software Sustainability Institute's report into the study observes "a clear demand and need to provide

software training, recognition and resourcing for software skills and techniques in the arts and humanities" [106].

3.2.1 A "Chasm" in Software Engineering Capability

The idea of a "chasm" between industry and academic approaches to software engineering was introduced by Kelly, who argued that with no formal training or education in software practices, scientists learn about software engineering from each other rather than from professional practitioners, leading to a perceived "chasm" between the quality of scientific and commercial software [107]. She asked for leaders and practitioners on both sides of the chasm to come together and define domain-specific software engineering methodologies. In that sense, her position was the opposite of later advocates of the "chasm" metaphor, who saw the chasm as a capability gap that should be "bridged" by scientific programmers adopting industry practices, led by Storer [108].

Johanson and Hasselbring describe the relationship between software engineering and computational science as being one of "isolation" that has led to a "productivity and credibility crisis of computational science" [109, §1]. They observe (*op. cit.*, [§4.1]) that existing chasm-bridging approaches—publishing source code with scientific articles, involving software engineers in research software construction, and training scientists in software engineering methods—have failed to address the crisis they identify. They identify integrating domain-specific languages (DSLs) into scientific computing workflows, adopting software performance engineering and software testing techniques, and an iterative approach to software requirements management, as potential future approaches to improving scientific software.

Storey and Baskerville identify scalability issues as a direct result of the software chasm; problems arise as applications become used by larger scientific communities [110]. They suggest that design practices, guided by design science research, can guide the creation of appropriate digital artefacts to support the increasing digitalisation of science.

3.2.2 Opinion and Thought Leadership

Much of the writing on how software in the research domain is—or should be—constructed is in opinion papers, and editorials in computing magazines such as *Communications of the* ACM. Wilson and Dunne wrote that the "bottleneck between our ears" is the problem, and that people working in computational science would do well to adopt some tools and techniques from commercial software engineering [111].

The lack of systematic analysis of software engineering among (industry) professionals, low engagement by practitioners with the scholarly literature, and high prevalence of argumentation by case presentation and analogy, means that much development in the field is trend- and opinion-driven. The state of the art in research software engineering is therefore similarly opinion-driven, especially if the guiding principle is chasm-bridging.

In this spirit, authors recommend practices that RSEs should adopt as best practice [112] [113], good enough practice [114], or barely sufficient practice [115].[1] These recommendations are based on perceived successful adoption in industry along with argumentation about aspects of research software that will be improved by their adoption among the RSE community. As argued in section 3.5 and demonstrated for the specific practice of Test-Driven Development, there can be a wide gulf between what is empirically known about software engineering practice, and the recommendations advocated by industry leaders.

The Turing Way is a community-driven manual of practices in data science, which includes recommendations for software development [116]. As with the opinion papers on software engineering practice, the recommendations in the Turing Way come from personal experience and logical argument, without either evidential or theoretical backing. Discussions at RSE conferences similarly depend on personal experience and opinion, rather than critical appraisal of evidence, supporting the craft-like nature of software engineering knowledge. Panel sessions at SeptembRSE, the online conference of the Society of Research Software Engineering, illustrate

[1] I am the lead author of [115].

this. Sessions on whether testing is "overkill for most research software" [117][2] and whether industry practices work in academia [118] cover the opinions and experiences of the panelists but do not rely on empirical observation or scholarly investigation. There is some limited scholarly research of research software engineering, particularly in the field of scientific computing; that literature is explored in the remaining parts of this section.

Research software engineers and their leadership do not appear to generate or make use of this literature in their writing on the RSE occupation. Interviews with RSEs can uncover whether they are in fact using the results from the scientific computing literature in synthesising their opinions, whether other factors stop them from spending time with this literature, or whether they are aware of it but do not find it relevant to their work. Field studies can additionally reveal the "tacit" engagement between RSEs and the scientific computing literature, uncovering instances in which engineers do read, review, or share papers with colleagues but leave no written record of doing so and don't explicitly mention these activities in interviews.

3.2.3 Open Source Software in Research

Open Source Software (OSS) is software that satisfies the Open Source Definition, requiring that the software be available to use for any purpose; to study and understand how it works; to modify; and to share along with any improvements someone makes [119]. Open Source components can be integrated into research software as an alternative to buying commercial software, contracting development, or building the component in-house [120].

Additionally, publishing research software as OSS satisfies expectations that the software be available for other researchers to use in replicating the results of publications that use the software (see section 2.4.6) and enables researchers and RSEs to build a community around using and developing the software. Neil Chue Hong points to the importance of a clear licence—possibly an OSS licence

[2] I was one of the participants in this panel session.

identified using a Software Package Data Exchange (SPDX) code—in making research software reusable [121].

Fortunato and Galassi make a case for OSS enabling reproducible research within an ecosystem of open scholarship [122]. Lee[3] describes publishing software in the Journal of Open Source Software as a method to generate a Digital Object Identifier (DOI) that helps other researchers to cite the software in their publications [123].

Hanwell et al. share a case study of the growth of a sustainable open science community around their open source Visual Toolkit software [124]. Conversely, Muna et al. describe the problems of developing infrastructural research software as an open source project: a large dependent community making requests on the contributors' time, a lack of funding, and lack of recognition for contributors in the academic ecosystem [125].

3.2.4 Quantitative Research into Scientific Computing

Few quantitative studies of scientific computing practised in the field (as distinct from software engineering lab experiments in which the participants happen to be students of computer science or software engineering) exist.

In researching the importance of software construction to scientific research, Wiese et al. asked a survey of R programmers who self-identified as scientists 'What are the three most pressing problems, challenges, issues, irritations, or other "pain points" you encounter when developing scientific software?' [126] They grouped reported pain points into three high-level categories: technical problems, social problems, and scientific-related problems.

The authors generate Ishikawa (fishbone) diagrams of each of these three categories, indicating how they coded the pain points into sub-categories and the relative prevalence of each pain point (figures 1-3 in [126]). The most popularly-reported problems were technical problems (approximately 70% of the total responses), of which the most prevalent were problems related to requirements management. Of these, documentation issues were reported most frequently.

[3] No relation to this author.

The sampling approach used means that the respondents all use R, and all have published at least one module to the Comprehensive R Archive Network (from which we infer that they understand modularity in software design to some level), which may not reflect the experience of all practitioners in scientific computing. It is also possible that the categorisation and coding that the authors came up with does not match the way in which participants relate these issues: perhaps programmers see documentation as a social rather than a technical issue, for example. Nonetheless the "pain points" they identify across a broad sub-population of scientific programmers could be compared with those discovered in different samples, and used as the basis for exploratory questions in self-structured interviews to find out how RSE practice ameliorates or exacerbates these pains.

Sletholt et al. identified 35 practices within the Extreme Programming (XP) and Scrum methodologies that they classify as agile practices, including project management practices (practice 1, "Priorities (Product Backlog) maintained by a dedicated role (Product Owner)"), programming practices (practice 20, "Code the unit test first"), and testing practices (practice 34, "When a bug is found tests are created") [127]. Their method is a literature review of scientific software case reports, of the type described in more detail below in section 3.2.5. They discovered that some of the practices they identified were common across many or all projects studied (for example practice 12, "Release planning to release product increments"), some were mostly mentioned on projects that explicitly do not adopt the practice (for example practice 21, "All production code is pair programmed"), and some were not mentioned in any of the reports they investigated (for example practice 28, "Use CRC cards for design sessions"[4]).

The authors conclude, based on the description in the experience reports, that all papers indicated positive effects of agile practices where adopted. However, many of the papers mention just a few practices, and all but one of them explicitly rule out adoption of some of the agile practices. This indicates an "à la carte" approach to practice adoption, which could be due to lack of awareness of the

[4]Class-Responsibility-Collaborator (CRC) cards are index cards on which software engineers capture important information about the design of classes in object-oriented programming [128].

practices; immaturity of the researchers' processes and practice adoption; or a lack of relevance of the practices to the research context. Or the researchers may have a very mature and evolving software development lifecycle (for example, at maturity level 5 in the CMM–I [129]) that transcends the prescriptions of any methodology.

3.2.5 Experience Reports from Research Software Projects

Case studies and experience reports from particular scientific computation projects reveal how the project participants developed their software, and the opinions they have of how it went. Segal argues that while empirical software engineering is traditionally light on case reports, they are an essential part of the research landscape as they describe how software engineering is actually practised in the field and can thus connect theoretical ideas to the practical reality [130].

As summarised in [127], a number of the case studies in RSE report on the social side of the projects—how they are organised and managed, rather than the technical side—how the software is written.

Katz et al. compare the operation of a "traditional" RSE group in the University of Manchester with two groups based in United States institutions, at University of Illinois at Urbana-Champaign and University of Notre Dame, respectively [131]. During this "expansion" phase of RSE, in which institutions are forming new groups and hiring new RSEs, such experience reports provide useful insight into the governance of academic software engineering using data that is unlikely to leave a trace in software repositories, bug databases or other public archival sources frequently used in quantitative software engineering analysis. The authors' list of positive and negative aspects of the group structures (Table 1 in [131]) provide useful starting points for further investigation of how RSEs work. For example, all teams rely on "institutional memory" via experienced staff who remain with the teams between projects (Katz et al. contrast this situation with post-doc researchers on short contracts who are likely to find other contracts and move on); the interaction between "institutional memory" and explicit documentation among creators of research software is unexplored in the literature.

In a case study involving a team developing a lab-based software system for use in an in-space instrument, Segal reports that research scientists were not comfortable with the level of requirements specification and documentation expected by the software developers, and that the documentation failed to create a shared understanding between both groups [132]. They propose interventions for that project, including the XP practice of pair programming. Such recommendations would be appropriate for similar projects, and it would be useful to understand the variety of projects and practices throughout RSE to know whether such recommendations are generalisable across the field.

3.2.6 The Gap in RSE Research

There is not much scholarly literature on the work of research software engineers, and that which exists does not provide the holistic insight needed to generate a systemic understanding of the field that could motivate high-impact changes. The empirical studies that have been published adopt quantitative approaches, for example, surveys and literature reviews, leaving the qualitative analysis of who RSEs are (or who they believe themselves to be) and how and why they perform their work unexplored.

There are plenty of opinions published in the literature, for example the "chasm" and the lists of desirable practices we reported on in section 3.2.2, and these opinions are not supported by substantive research. We suggest that these opinions could reasonably be formulated as hypotheses about RSE that can be developed into research questions for empirical study.

As GT has been successfully used to explain organisation of and collaboration within teams of agile software developers in industry (see section 3.5), we think it would be a fruitful source of explicating theory for research software engineering. Hoda suggests that Grounded Theory (GT) could provide this understanding, by providing a theory of research software engineering derived from the practitioners and data that they generate. In her description of Socio-Technical Grounded Theory for Software Engineering, Hoda describes the "sweet spot" for GT in software engineering research as where:

- the research focuses on human and social aspects;
- the area lacks existing theories;
- the topic is practice-relevant;
- the questions are complex and deep;
- studies are primarily qualitative; and
- data are collected through interviews and observations. [133]

A search of SOLO[5] using the phrase `"grounded theory" AND "research software engineering"` conducted on September 1st 2021 yielded 38 peer-reviewed articles, but on investigation of their abstracts only one of these articles reports an application of GT to a research software context. That article developed a theory of scientific software training using blended learning [134], it does not investigate how research software is built.

We agree with Hoda that the criteria for considering GT are satisfied in RSE, but as both a broad and deep field we do not believe that developing a full theory in a doctoral research programme is feasible: it would be difficult to reach theoretical saturation with the resources and time available. We suggest that a mixed-methods study would provide useful insight in Hoda's "sweet spot" and serve as an initial basis for future research to develop a grounded theory.

The paucity of literature on RSE suggests that this is a field that has received very little academic attention. We therefore turn to related fields to examine work that could provide useful guidance on, or an analogy for, an investigation into RSE as a professional practice. We start with other community-led developments in academic culture; FAIR data management, and responsible research and innovation.

[5]The Bodleian libraries' search engine—Search Oxford Libraries Online https://solo.bodleian.ox.ac.uk.

3.3 FAIR Principles and Research Data Management

The development of the FAIR guiding principles for scientific data management provide an illuminating analogy to the development of RSE for research software. Like Research Software Engineering, the FAIR principles arose from a community discussion. Participants at a 2014 workshop, "Jointly Designing a Data Fairport", met to discuss overcoming obstacles in data discovery and reuse [99]. The resulting principles of Findability, Accessibility, Interoperability, and Reusability were created to inform, but not dictate, data management practice and connect it to academic values.

According to Wilkinson et al. [99], the goal is "for scholarly digital objects of all kinds to become 'first class citizens' in the scientific publication ecosystem". This goal is an imperfect analogy to the motivation of the group who met at the Software Sustainability Institute's Collaborations Workshop in 2012 to discuss the lack of recognition and career pathway for software engineers contributing to research [2], though in both cases practitioners observed a gap in research practice and motivation, and acted to highlight that gap and the impact it has on the research being undertaken.

Wilkinson et al. foreground the relevance and benefit that systematic approaches to research data management bring to a broad collection of stakeholders [99]. They include 'computational stakeholders'—autonomous software systems that potentially encounter new data sets and incorporate them into automatic analysis. This ecosystem is comparable with the stakeholders identified for research software in section 2.2.3; note the absence of 'computational stakeholders' identified for research software.

The need for accessible, interoperable, and reusable data curation was identified by epidemiologists as crucial to research on the SARS-CoV-2 virus and Covid-19 pandemic [63]. In a presentation to the United Nations World Data Forum, Stefaan Verhulst of the Data Stewards Network identifies a set of skills that form the occupation of "data steward" [135]. According to Verhulst, data stewards are

"professionals empowered to create public value (including official statistics) by re-using data and public expertise [who] form a new—and essential—link in the data value chain". This development of a new professional role of data steward parallels the development of the Research Software Engineer profession, and could provide useful analogies and case studies.

In addition to principled arguments in support of FAIR, arguments have been made in support of a financial cost due to redundant funding of repeated work, research being retracted, multiple copies of data being stored to work around access limits, and other factors. A European Commission report found that the direct cost of non-FAIR data management to the EU economy was €10.2 billion, with another €16 billion estimated indirect costs [136]. These results point to the possibility of analogous analysis for RSE using data from funding organisations.

There have also been multiple attempts to define FAIR for research software. Hasselbring et al. argue that research software is an artefact that is as important to the reusability and reproducibility of computational research as data is, and therefore that the same principles should apply to the management of research software as to that of research data [102]. They further say that research software engineering is needed to reliably imbue the software creation process with these principles. A "FAIR 4 Research Software" working group work to translate the FAIR principles to software management [103]. This all suggests that FAIR may turn up in the work of RSEs in two forms: the FAIRness of data generated, transformed, or consumed by the software; and the software itself as a research artefact.

3.4 Responsible Research and Innovation

Another recent development of a community-led culture shift in academia, analogous to that attempted in RSE, is Responsible Research and Innovation (RRI).

As we will discuss in depth in section 3.6.1, one of the defining characteristics of a profession is the normative value of service to others. In computing, professional organisations including the ACM have attempted to codify such values in codes of professional conduct [137] but these efforts have not seen wide enough adoption to be

considered normative. The author of this thesis noted in a previous publication that the worldwide membership of the ACM, for example, is less than half the number of people employed in computing occupations in the United States alone [54].

Rather than promoting adherence to a written code of practice, RRI suggests a collaborative and inclusive framework of interaction in which input from diverse relevant stakeholders is solicited and used throughout the research process from initial design through to development and deployment of products or other innovations resulting from the research [97].

The need for a more reflexive and inclusive approach to research governance was first identified in synthetic biology [138], a field in which research can find dual-use translations, unanticipated consequences, and societal concerns that are not taken into account in traditional research governence. The ideas were adopted and developed in nanotechnology [139] and in geo-engineering [140], becoming known as *responsible innovation*. There are four categories that a responsible approach to innovation governance addresses [141]:

- **Anticipation.** Identifying and imagining plausible future scenarios involving the innovation, and their consequences.
- **Reflexivity.** Understanding the researchers' own responsibilities, moral and ethical contributions, and role in innovation governance.
- **Inclusion.** Ensuring that research governance includes a diverse representation of stakeholders and acknowledges social stakes and power imbalances in relationships affected by the innovation.
- **Responsiveness.** Being open and reactive to issues raised in relation to the innovation.

RRI has also been applied to information and communication technology (ICT), and in particular software development. It is argued by researchers in the field of computing and social responsibility that RRI should replace the existing idea of "computer ethics", exemplified in the joint IEEE-ACM code of practice described in

section 2.3.5, as the "reference discourse" in responsible application of computing and information technology development [98]. But this is not a simple matter of replacing one set of rules and tools with another. Jirotka et al. point to the necessity "to understand how ICT researchers and practitioners manage their professional responsibilities, as well as how they perceive the notion of RRI" as a challenge to further embedding of RRI into ICT innovation [97].

Stahl et al. note that ICT already involves stakeholders in innovation design, particularly through requirements engineering and human-computer interaction practices including workshops, scenario development, and focus groups that can directly translate into RRI practices. However, they also indicate changes to role definitions and problem-framing needed for researchers in the field to adopt a reflexive, anticipatory approach to governance, including a reluctance to anticipate future use of innovations, and an assumption that researchers are moral actors by default [142].

Two examples from the Stati Generali dell'Innovazione (SGI)—an Italian not-for-profit association formed with the goal of promoting collaboration between scientists, politicians, citizens and other actors to create a shared perspective on innovations—demonstrate the effect of RRI thinking on development of computing innovations.

The SGI's focus on Open Science, including open access to data and open source software, connects the association's work with questions of responsible innovation [143], as well as the principles of FAIR data and FAIR for research software discussed in section 3.3. The authors of [143] particularly highlight the conflicting needs of patient expectations of confidentiality of their health records, with "big data" analyses requiring access to data in the aggregate and inter-organisational collaboration on public health, and of wide access to medical interventions (such as vaccines) conflicting with intellectual property restrictions imposed by inventors.

Another project at SGI explored the responsible development of innovations within an Open Government model related to geospatial technology [144]. The project incorporates the expectations of innovators with public administrators and residents in so-called "smart cities" that use geospatial information and internet of

things (IoT) sensors in their governance. They define "territorial prosumerism", a view of citizens based on mutuality in which the data produced by the citizens to enable smart city innovations is also available for those citizens to consume.

A UK-created model of an RRI approach has been adopted as guidance for researchers funded by the Engineering and Physical Sciences Research Council in the United Kingdom as the AREA framework [145]. Different models are used in other regions, for example the six-dimensional approach selected for the European Union's Horizon 2020 programme [146].

3.5 Scholarly Investigation and Research of Software Engineering

There is a large amount of scholarly interest in software engineering. Much of it is basic research into engineering tools such as correctness provers or test generators, which while useful for advancing the state of the art do not provide insight into the everyday practices of working software engineers. In this section we focus on literature related to software engineering practice and organisation to understand the approaches used, the results obtained, and whether these results are ecologically valid in RSE or could suggest the application of similar methods in that field.

3.5.1 Quantitative Research in Software Engineering

Common methods in software engineering research include quantitative data analysis by data mining, or by surveying practitioners. Both of these techniques allow large-scale data gathering and therefore a good overview of the current state of practice, but their results are not necessarily ecologically valid in a particular context and they can both miss important information.

Data-Mining Studies

One example of the limitations of a data-mining methodology is Borle et al.'s study of Test-Driven Development (TDD) [147]. We choose this example because Greg Wilson recently described it as one of a number of empirical software engineering

papers that found no effect when developers use TDD, saying "until we see a paper indicating that TDD does work, we're going to set this topic aside" [148].

The authors of [147] mined information from git repositories hosted on GitHub to find out whether TDD was employed, and then compared various dependent variables between TDD and non-TDD repositories to determine whether the practice of Test-Driven Development has any impact on software outcomes. They note the construct validity problem that the commit history of a git repository does not contain enough information to determine whether the programmers used TDD in their work, and their approximation is to look for test files with the same names as implementation files committed at roughly the same time.

Unfortunately the dependent variables in [147] are also questionably constructed. One such variable, the commit velocity, is defined as the average length of time between commits. Borle et al. do not control for the time allocation developers allotted to their projects, so a repository with low commit velocity could be the result of hobbyist developers not dedicating much time to the project or the result of slow processes. Another variable, the bug fix rate, is supposed to measure how many commits fix reported problems but uses the proxy metric of whether the commit message contains a message like "fixes bug 1234". This approach is subject to both false positives (a developer writes that a commit fixes a bug when it doesn't) and false negatives (a developer doesn't record that they fixed a bug in a commit).

Even were this data-mining study to reliably compare the outcomes of software development efforts in situations where the developers do or do not employ TDD, there would still be many important aspects of process selection that it cannot address. There may be a cultural or social motivation (whether a benefit or a pressure) to the choice to adopt TDD. The adoption—or avoidance—of the practice might affect an engineer's *opinion* of their work or its quality. We also do not know whether the engineers whose projects were assessed to be TDD-using projects agree that they adopted TDD.

Quantitative Experiments

A second study Wilson has highlighted for its negative result investigating benefits from TDD is Fucci et al.'s comparison of performance when engineers use test-first or test-last development [149] [150]. This study used a crossover experiment design, in which 21 conveniently-sampled graduate students were given two programming tasks to complete, one using test-first development and one using test-last development. Each task was performed during a three-hour lab session. The students were randomly assigned the order in which to use development approaches; the order in which they completed the tasks was kept constant.

The authors found no significant correlation between the testing approach and any of their dependent variables: the number of tests written; the external code quality; or the developers' productivity. They did observe an ordering effect but determined that this was not due to crossover (i.e. experience in the first condition informing behaviour in the second).

As Fauci et al. note in the threats to validity of [149], the recruitment of a homogenous group of student participators challenges the ecological validity of the results in industry. Similarly, there could be different effects in a three-hour lab session than when a professional practitioner applies a practice consistently through the duration of their career. Another problem with this study is that it does not include the "no testing at all" condition common in industry. Regardless, the repeated lack of evidence in support of TDD ([149] is a replication of an earlier study, which is itself a replication) should be cause to question the advocacy-based discussion of practice common in industry: "the bottom line is that TDD works, and everybody needs to get over it" [151]. It is interesting that the industry thought leader Robert Martin reaches the exact opposite conclusion regarding the efficacy of this practice using his commercial experience than the academic Greg Wilson derives from the empirical software engineering literature.

Survey-Based Research

Aniche and Gerosa conducted a survey of TDD practitioners to discover what programmers think about the benefits of the practice, and whether they commonly make mistakes when applying the test-driven approach. Their convenience sample of 218 practitioners (90% of whom apply TDD in industry) discovered that 60% of programmers believe that TDD results in lower defect density, and that 60% believe that adopting the practice leads to higher-quality code. They also discovered that the "mistakes" they asked programmers about occur frequently; for example 44% of "experienced" (more than 4 years) programmers regularly or frequently forget the refactoring step[6] as do 52% of beginners.

It would be fruitful to find out whether these are genuine mistakes, conscious decisions by practitioners to take practical shortcuts, or tacit changes to their practice. Such research could distinguish between inadequate training or reinforcement of TDD among developers and undocumented innovation in practice developed through experience. The Aniche and Gerosa survey also raises the question of what properties developers are considering when they say that test-driven code is "higher quality". Taken together, these studies of test-driven development demonstrate the limitations of purely quantitative approaches to understanding software engineering practice. Further information must be gathered to understand the behaviours of software engineers, whether particular practices provide benefits that do not show up in software artifacts, and whether lapses or limited uptake in practice are due to lack of awareness or education, or deliberate contextual choice.

Hannay et al. devised a questionnaire to survey scientists on their knowledge of and comfort with software development techniques, and on where they learned those skills [105]. The survey questions used are not included in the paper, nor as supplementary material. Any observations on the designs of the questions must be inferred from the conclusions the authors draw from their data, and limited information about the questions included in the published script.

[6] TDD is summarised with the mantra "red, green, refactor": write a failing test (red), make it pass (green), improve the code design without re-introducing a failure (refactor).

Hannay et al. report advertising the survey in multiple locations, including an advert in the journal *American Scientist*.[7] They thus received a sample of nearly 2,000 respondents from a variety of demographic backgrounds, roles, and career stages but with a bias towards Europe and North America. It is not clear whether this reflects the population distribution of the scientific community or a limitation caused by the way in which they recruited participants.

Many of the questions asked participants directly about their opinions on the importance of different aspects of software development to their scientific work. Some of these questions are reported as using a five-point scale "('not at all important', 'not important', 'somewhat important', 'important', 'very important')", with the results summarised and hypotheses tested by measuring the fraction of respondents who answered "important" or "very important" to those questions. These questions are subject to acquiescence bias, in that respondents can agree with the researchers that certain aspects of software development must indeed be important, as the researchers thought to ask about them in their questionnaire.

The survey measures perceptions of software development by scientists without correlation to experience or evidence. For example, the authors find that 96.9% of respondents report that informal self-study is important or very important for developing software, but do not explore how much time respondents allow for informal self-study or other means of acquiring software development knowledge. Similarly, only 17.1% of respondents *believe* that formal training at work is important, but we do not know how many have access to or have *taken* formal training.

Respondents are asked to rank eight different software engineering concepts by both importance and their understanding of the concepts. It can be imagined that these measures would be correlated, as people who see a concept as unimportant would not be motivated to deepen their understanding of it, and those who do not understand a concept might downplay its importance to cover for the gap in their knowledge. Also, the work of Kruger and Dunning discovered that people with low ability at a skill tend to overreport their self-assessed capability [152]; the

[7] The page containing the advertisement was not available online, and conditions related to the COVID-19 pandemic at time of writing made it impractical to access the print edition.

responses about understanding of software engineering concepts in [105] should be treated as systematically biased but the authors do not provide any external calibration of the results.

In general, the conclusions drawn in [105] are stated to be about software development in scientific computing, but instead explore the related issue of the perception scientists have of software development and how they perform it. For example, the above-quoted numbers of respondents who believe that informal self-study is important or very important for developing software are used to support the authors' hypothesis H1a "Most scientists learn most of what they know about *developing* software on their own or informally from their peers, rather than through formal training" (emphasis original). The question of whether scientists believe self-study is important cannot yield data in support of whether self-study is undertaken.

Pinto et al. replicated the survey from [105] a decade later, in [153]. They report receiving the original set of questions used in [105] from those authors, and using the same questions in their replication. The limitations identified in section 3.5.1 are therefore applicable to this paper too, but on the whole reusing the same questions was a good idea to permit comparison between the results of the two surveys.

The replication authors recruited scientists who had published at least one package to the Comprehensive R Archive Network (CRAN). The sample they obtained is thus constituted of scientists who use R and are comfortable enough with ideas of packaging and software reuse to have published one or more packages to a public archive. Pinto et al. treat the comparison of their results with [105] as a simple longitudinal comparison, but the different sampling methodologies add another factor to the situation.

Pinto et al. asked some additional questions which were not reported in Hannay et al.'s original survey, which provide some of the experiential evidence missing from that paper.[8] For example, they ask respondents to indicate how frequently they perform various software engineering activities, which can be compared with the "importance" respondents place on each of those activities.

[8]The full survey, along with the data mining scripts used and instructions on replicating their analysis, are provided at https://github.com/fronchetti/SANER-2017.

As discussed in section 3.2.4, they also asked respondents a free-form question about the "pain points" they encountered in software development. This type of question requires more effort to analyse than categorical questions, and the authors wrote a second paper explaining their approach to coding the answers and the results they found [126]. This question provides more useful context for the rest of the survey, by illuminating the specific problems that scientists have with software development which could yield curricula for training interventions, or topics for community documentation.

Reggio et al. surveyed practitioners of software engineering in the Internet of Things (IoT) domain to "[understand] the nature of IoT systems and the vital aspects of their design from a practical point of view" [154]. Their questionnaire was posted online and a combination of convenience and snowball sampling via social networks used to find respondents. The questions and raw data are available at `https://sepl.dibris.unige.it/2020-IoT-Survey.php`. The authors acknowledge that they rely on respondents' self-reports of their IoT experience to choose whether to include their responses in the data set and that this means the survey may include fake responses.

The design of the survey is a good fit with the research goal. Respondents are asked to describe various characteristics of their use of IoT, how various software engineering attributes were relevant to the projects they have worked on, and how challenging they found meeting the desired level of quality in those attributes.

Where the researchers asked respondents to define the important elements and characteristics of IoT systems, and types of IoT systems, they provided a collection of answers for respondents to choose from and an "other" field where they could write free text. Few respondents (12% for the elements and characteristics question, 5% for the types of systems question) wrote their own suggestions in "other", from which the authors conclude that their own categorisation was "sufficiently complete". An alternative explanation is that respondents didn't feel the additional effort required to think up and write in their own answers was warranted.

3.5.2 Qualitative Investigations of Software Engineering

Qualitative research, particularly on the behavioural factors influencing software engineering, is rare. Some early luminaries in the software engineering field wrote about their experiences of the personal and interpersonal issues they encountered as managers or consultants (for example Brooks [155], Weinberg [156], and DeMarco & Lister [157]), but systematic studies in the scholarly literature are few.

The editors of an IEEE Software special edition on the "behavioural science of software engineering" reported that much empirical research on software engineering "remains primarily focused on the technical aspects of the tools and processes, without considering humans in [the researchers'] evaluations" [158]. In a review of 151 papers from the International Conference of Software Engineering and journal of Empirical Software Engineering, Storey et al. discovered that the majority (57%) of papers are solution-oriented rather than descriptive of the current state of practice, and that a majority (59%) depend on data mined from project repositories and archives [159]. Only 7% of the papers they surveyed derived results from field observations.

From a philosophical perspective, Dittrich argues that there is a missing "theory of practice" in software engineering [160]. A software engineering method (for example, Scrum) encodes a perspective about the purpose of the engineering activity and the composition and values of the system and its context; principles about the organisation of the development and process; and tools and techniques that support the practice of completing tasks that contribute to the development. In the field, software engineers often adapt or ignore the prescriptions of the method when they perceive contextual needs to "work around" the method in the field, with only 6% of practitioners applying a defined method ([161], cited in [160]). Therefore a theory of RSE with explaining power needs to be grounded in the actual practice of RSEs, which may differ (even if tacitly) from the methods those engineers claim to adopt.

3.5.3 Grounded Theory Research in Software Engineering

Grounded theory (GT) is rooted in the work of Barney Glaser and Anselm Strauss [162], although today three distinct "strands" of GT are recognised [163]. The first,

often called "classic" or "Glaserian" GT, adopts an epistemology that some scholars view as objectivist while others see it as post-positivist [163, p.3]: reality exists and the researcher's job is to discover and describe it. In Glaserian GT, the researcher takes a completely neutral approach to the research domain, with no literature review conducted *a priori* and no predetermined thoughts. They engage with the domain as detached observers, although they do check the categorisation in their coding for "emergent fit" with existing theories from the literature as analysis progresses.

Glaserian GT uses two phases of data coding to generate the theory.

1. *Substantive coding*, in which the researcher categorises the data and identifies the core category for the theory.

2. *Theoretical coding*, in which they connect the categories to derive the grounded theory.

While Strauss worked with Glaser to originally describe GT, his later work with Juliet Corbin represented a departure from classic GT that Glaser did not accept, and led to a second, "Straussian" strand of the methodology. Straussian GT adopts a symbolic interactionist epistemology rooted in pragmatism [163, p.3]: while an objective reality exists, any person's perception of it is constructed through their interactions and interpretations, and understanding these is important to developing an understanding of human behaviour. The researcher is no longer neutral, as their interactions with the participants in the study and their existing experiences shape their interpretation of the data and the hypotheses they construct.

Where a Glaserian researcher conducting an interview would take field notes during the interview and rely on those for later analysis, detaching the participant's voice from the data generated, in Straussian GT a researcher can record or transcribe the interview and continuously refer back to their conversation with the participant as they refine their analysis and generate their theory [164, p.55-56].

Straussian GT offers a more detailed recipe for practice than Glaserian GT, and introduces three coding phases.

1. *Open coding*, in which the researcher codes the text, discovering and dimensionalising categories [163, p.7].

2. *Axial coding*, in which they use a coding paradigm to group categories based on emotions, inter/actions, and consequences.

3. *Selective coding* (or theoretical integration), in which they identify a core category and relate the other categories to the core category to build the grounded theory.

Finally, Kathy Charmaz introduced a third, "constructivist" strand of GT, which replatforms grounded theory on a constructivist epistemology [163, p.7]. In this subjective paradigm, whether the researcher accepts the existence of an objective reality or not, the socially constructed and perceived reality that people negotiate in their day-to-day lives is the important target of study. The researcher and their participants co-create the grounded theory, as the researcher is fully embedded in the conversations and interactions that lead to the theory generation.

Constructivist GT is less prescriptive than either of the other two strands: researchers can use existing theories and literature as sensitising concepts to guide their research, and go from initial coding of their data to categorisation and theory generation, though they can adopt any of the techniques from the other strands that support their analysis [165]. Researchers make heavy use of memoing as a tool to understand their own place in the research and to ensure that their preconceived ideas are not forced onto the data.

Stol et al. consider Grounded Theory (GT) to be a useful method for exploring and understanding software engineering, due to the relative novelty of the field and the lack of a theoretical framework describing the social, cultural, and human aspects of the practice [166]. Their 2016 survey of the literature uncovered 98 articles that mentioned GT, of which only eight[9] generated a theory from their use

[9]Stol et al. report that "a ninth article presented a set of hypotheses that could be considered a theory."

of techniques in the GT literature. This suggests that GT is still lightly used in software engineering research and can provide useful contributions to the field.

One of the principles behind the manifesto for Agile Software Development is that "the best architectures, requirements, and designs emerge from self-organising teams" [42]. Hoda et al. developed a grounded theory to explain self-organisation in Agile teams. Their theory explains that self-organising teams perform a "balancing act" along three different dimensions: balancing freedom and responsibility; cross-functionalism and specialisation; and balancing continuous learning and iteration pressure[167].

Martin et al. developed a grounded theory to explore how teams following XP implemented the practice "the customer is always available" [168]. They discovered that there are many tasks for the customer to perform and responsibilities they must fulfil. As such, while the XP practice can be interpreted as requiring the team to contain a single customer representative or proxy, often a "customer team" is formed to share the workload [169]. This form of insight into the social negotiation of software projects depends on observation of how people really work irrespective of any documented or desired processes and methodologies, and would be difficult or impossible to uncover using purely deductive, quantitative approaches.

3.6 The Sociology of Professions and Professionalism

The development of sociological theories of what professions are and how they are formed and maintained provides a theoretical basis for the definition of a profession and the process of its development, which can be applied to an understanding of RSE. It also demonstrates research methods that have been successful in the study of other professions and can be implemented in researching RSE.

Trends in the development of social science research into professions—particularly the turn away from general categorisation and theory toward the study of specific occupations and their practitioners—are seen in the field's review literature. Researchers in the sociology of professions agree on the development of three distinct

"schools" in the field: the "functionalist" school in the mid-twentieth century; the "interactionist" school in the 1960s-1980s; and a "social conflict" school developing at the turn of the twenty-first century [170] [171].

3.6.1 What is a Profession?

The functionalist approach coincided with a "golden age" of the traditional professions (law, medicine, academia, engineering, and others), and sought to identify the features that distinguished a profession from a non-professionalised occupation: expert knowledge; technical autonomy; a normative value of service toward clients and society[10], and high status in society [171].

Organisations promoting software engineering or computing as a profession can be seen to develop these explicit "features" of a profession, as with the Association for Computing Machinery's work on an accredited curriculum for degree-level computing [172], a software engineering body of knowledge [33], and a professional code of practice [137]. How the specific field of RSE collectively engages with these materials, or produces its own professional trappings, is an open question; researching this question would help to produce a "manifesto" for RSE that is sensitive to the expectations and norms of the current state of practice.

3.6.2 How do Professions Develop?

In the interactionist school, sociologists developed causal theories of the "professionalisation" of occupations, understanding how occupations arise from technological or social changes and how practitioners develop (or fail to develop) prestige, exclusivity, and the other trappings of a profession [170]. Dingwall reported how the occupation of health visiting arose from voluntary sanitary associations, became a distinct (and paid) occupation in its own right, and ultimately was absorbed into the nursing profession [173]. His model includes both the "fission" of new occupations from existing ones, when the labour involved is perceived as particularly prestigious or "dirty", or technological changes alter the division of labour, and also the

[10] Two sub-schools differ on whether professions are genuinely altruistic (Durkheimian) or whether this outward presentation of service is in the professionals' self-interest (Weberian).

"fusion" of multiple roles as existing professional organisations claim jurisdiction over the occupation.

The jurisdictional conflict and ideas of fission and fusion of occupations can be seen in professionalisation activities in software engineering. Dedicated computing organisations clashed or collaborated with engineering institutions in multiple countries, over the scope of a licensed computing profession and whether software engineering is a branch of engineering or a distinct role [32] [174]. Research Software Engineering inherits this intersection with engineering, and additionally introduces other potential jurisdictional conflicts with academia: people working on research software at a research-producing organisation could be seen as researchers who produce software; as software creators in research or academic roles; or as professional services staff. Management practices in research-producing organisations are not always aligned with employee well-being, in either academic or professional services roles, with control governance practices being beneficial to professional services staff and detrimental to non-leadership academic staff [175], so the identification of RSE as either an academic-related or professional service role has consequences for the relationship between the practitioners and their hiring institutions. Whitchurch notes that the interface between professional and academic activity is blurred, with a rise of "third space" professionals that—on their own initiative or at their institutions' appointment—cross or even entirely disregard structural boundaries [176].

There can also be jurisidictional conflict with institutions promoting computing and software engineering, as RSE could be seen as a distinct role that needs its own advocacy, or as a "branch" of software engineering. These jurisdictional issues highlight potential conflicts, and strategies to mitigate those conflicts, for RSE leadership and establish the extent to which career development and pathways in RSE are informed by, and distinct from, its neighbouring and intersecting professions.

3.6.3 How Professionals Work

Towards the end of the twentieth century, the growth of multinational corporations in a globalised economy and the tendency for service workers to be employed by

those corporations rather than engaged as independent agents reduced the impetus for new professions, and along with it reduced the theoretical development of the sociology of professions [170]. Social scientists turned to understanding the organisation and performance of knowledge work [171], frequently being hosted by software engineers in their fieldwork. The research into the behaviour of software engineering professionals (and those they collaborate with) yields specific insights into the behaviour of software engineers in industry, and could be particularly relevant to understanding the work of RSEs. Questions about whether the same behaviour arises in RSE, i.e. whether the conclusions are ecologically valid in academia, are relevant to understanding whether the approach identified by [111] of borrowing more commercial practices into research software and named "bridging the chasm" by [108] and described above are likely to succeed, or whether the argument in [107] that research software is a distinct context requiring contextually-relevant methods is valid.

Perlow explored the "time famine" of software engineers, and how their own attitudes along with management incentives reinforced a vicious circle of long hours of inefficient work that still led to missed deadlines [177]. Software engineers hosting Perlow saw solo working time as the "real" work and collaboration and meetings as annoying distractions, but as a result would often have to interrupt each other to make progress as they had no structured communication systems.

This ethnography was conducted in 1999, before the widespread adoption of Agile Software Development with its explicit value of "individuals and interactions over processes and tools" and principle that "business people and developers must work together daily throughout the project." [42] It would be informative to compare this "time famine" fieldwork with a team following an Agile methodology; additionally RSEs in academic institutions will have to navigate the expectation of long working hours in academia [178] and that interplay could affect how tasks are allocated and completed in RSE groups.

More recently, Kameo studied software engineers adopting the Scrum methodology in an organisation, and found that the long-term "organisation memory"

of management innovations being mandated from above only to be abandoned or replaced later, sometimes with significant negative impacts on the careers of the people who had most adapted to the innovations, led to a skepticism among practitioners and a desire to commit to the methodology only as much as they thought management were committed [179]. This perceived level of commitment came from "reading tea-leaves" of management behaviour, not from explicit statements regarding managers' intentions.

As Katz et al. argued for the benefit of "institutional memory" on research software engineering groups to ensure continuity of technical knowledge [131], it would be useful to understand how the benefits of such institutional memory interact with the cultural inflexibility observed in Kameo's organisation memory. Kameo observed a culture of skepticism to managerial innovations among software engineers, which might occur in RSE contexts.

3.6.4 How do Professionals Practice their Professions?

The questions asked above of Kameo's work [179]—whether interventions designed to address one particular observed problem would have negative consequences on another issue that was beyond the scope of the research—are generally applicable to studies like it and Perlow's study [177]. Such studies provide useful illumination of professionals' behaviour in relation to a single concept (management innovation, or time management, in these examples) but do not contextualise them in relation to each other, the other concerns professionals have, or the varying contexts in which the work is performed. Grounded Theory studies often provide more of this context, by surfacing the ideas and relationships important to the participants rather than exploring a specific idea chosen *a priori* by the researcher.

The healthcare professions have been studied by many researchers using grounded theory. While the results of these studies are very unlikely to translate directly to the RSE profession, examining this literature does show how a grounded theory approach can give broad insight into professional practice by centring the professionals' own views and experiences regarding their work.

Schreiber and MacDonald's study of nurse-anaesthetists in the United States demonstrates how a grounded theory approach can develop an explaining theory of an occupation at a general level, connecting individual practice with the socio-political context of the profession. They developed a theory in which practising nurse-anaesthetists "keep vigil over the patient" in which practitioners employ technical and social measures to ensure high-quality work, centring the importance of care of their patient [180]. Meanwhile nurse-anaesthetists also "keep vigil over the profession" by policing behaviour, maintaining credibility with the public and colleagues, and lobbying in situations where jurisdictional conflict with neighbouring professions occurs [181]. Connecting practical and political factors with an explaining theory would be of great benefit in RSE, where leaders and practitioners alike seek greater recognition for the value of the profession and definition for career development within it.

Calvert et al.'s grounded theory of midwifery in New Zealand explains that midwives are constantly "working towards being ready", with contextually relevant and available opportunities for professional development sought and engaged to ensure practitioners are always current in practice and able to deliver appropriate care [182]. Understanding how RSEs reflect on their capabilities, and how they engage with continuing professional development, is of vital importance in a field defined by perpetual technological change and a perceived capability "chasm" between practitioners in research and industry.

As we discussed in section 3.5, researchers have applied grounded theory to software engineering as a profession.

3.6.5 Neo-Institutional Theory

Neo-institutional theory (NIT) sees individual workers choosing their actions by weighing the competing institutions that press for their attention, and the logics that those institutions propose which connect their institutional structure, norms, and values with particular courses of action [183]. An institutional logic is a "way

of ordering time and space" that structures social order by imposing categories of meaning [184].

As a fictional example, consider a software developer working at a company who discovers a bug in the software they work on, while doing other work in the same module. They may be motivated to fix the bug and commit the change to their project's version control repository, from a sense of professional pride or valuing bug fixing as "doing the right thing". This normative sense of doing high-quality work could be learnt from a professional association, for example the ACM whose code of ethics asks computing professionals to "strive to achieve high quality in both the processes and products of professional work" [137]. However, their employer may expect the developer to follow a longer process, in which they create and submit a problem report, which then gets triaged and prioritised by others in the company. Later, if the bug gets selected to work on, it may get assigned to the developer to fix or to somebody else.

In this example, we see two organisations that propose competing logics, and the software developer must select from them. The ACM proposes striving to achieve high quality, which the developer interprets as meaning they should fix the bug that they discovered. The employer proposes following the problem report process.

Two works are considered seminal to NIT. In the first, Meyer & Rowan [185] introduced "formal structure as myth and ceremony". Rather than coordinating and controlling behaviour efficiently by adopting processes after rational consideration of their benefits, institutions create myths that formalise their organisational structures, and are adopted ceremonially if they are perceived to be legitimate. NIT moves beyond earlier, economist models of rational behaviour—and Marxian or Weberian theories predicated on particular macroscopic interactions between the state and other classes of actor—by introducing *rational myths*: widely-held but unverified assumptions or beliefs that particular actions are efficient or effective [186].

In our example of a developer choosing how to address a problem that they discovered in their software, the employer's process is a ceremony that embodies a rational myth justifying the roles that separately triage, prioritise, assign, and

fix problem reports. Equally, the ACM's ceremonies, legitimise the computing-as-profession myth and rationalise its code of ethics as a normative standard for workers in the field. Which logic the developer chooses to follow is based on their evaluation of the legitimacy of these logics and ceremonies.

Whether an actor accepts an institution's logic depends on the *legitimacy* of that institution's rational myths, so institutions' structures, ceremonies, and norms work to gain and reinforce institutional legitimacy. Suchman identifies three categories of legitimacy:

- *Pragmatic* legitimacy is based on self-interest, i.e. following the logic leads to perceived benefit for the actor;
- *Moral* legitimacy is based on normative approval, i.e. following the logic leads to acting within an accepted value structure; and
- *Cognitive* legitimacy is based on comprehensibility, i.e. following the logic is consistent with the actor's accepted understanding of the world and its organisation [187].

He further separates an organisation's actions to support its legitimacy into *institutional* legitimacy, which aligns with and reinforces dynamic social pressures but is outside the control of the organisation; and *strategic* legitimacy, in which organisation management deploys symbols that signal legitimacy.

In the second seminal work in NIT, DiMaggio & Powell [188] argue that organisations in a field (itself a categorisation that is imposed by institutional logic) undergo a process of *isomorphism*, in which change initiatives in distinct organisations have the paradoxical effect of making the organisations more alike even if they don't also make the organisations more efficient. They identify three processes that contribute to isomorphism:

- *Coercive* processes, in which formal pressures such as legislation, and informal pressures such as advocacy, act externally on organisations to bring about isomorphic changes.

- *Mimetic* processes, in which organisations facing challenges they are unprepared for duplicate observable attributes of organisations they perceive as successful.

- *Normative* processes, in which shared experiences and values, for example members of a protected profession with similar educational and social backgrounds and a shared professional network, lead to members of different organisations responding to their environments in similar ways.

Returning to our fictional example of the software developer, we would consider their company's problem-reporting process to be an example of coercive isomorphism if they operated in a legislated domain, and the regulations mandate that that process be followed; if their customers required them to follow the process to get their business; or if advocates working on behalf an institution like Scrum.org had convinced the company to adopt the process. We would consider the process to be an example of mimesis if, faced with the unfamiliar problem of handling problems in their software, the company turned to other organisations they perceived as successfully managing the problem and adopted what they saw of their processes. We would consider the process to be an example of normative isomorphism if the process had been designed by employees in the company who'd all seen the same process taught in university software engineering courses.

In his review of theories of professions, Mike Saks argues that NIT offers a benefit over other theories in that it doesn't presuppose particular features of the social organisation it studies: either the benevolent nature of professions expected in the functionalist model, or the class dynamics inherent in Marxist and Weberian analyses [189].

3.6.6 Communities of Practice

As described by one of its creators, Etienne Wenger, the theory of communities of practice is not exclusively a theory of professional work: anywhere that people identify themselves and each other through shared interest in activities, including

hobbies, political engagement, and education, can be the source of a community of practice, and Wenger cautions against either "encumbering the concept with too restrictive a definition" or "stretch[ing] the idea" [190, p.122]. Nevertheless, communities of practice has found applications in the study of professions, including as an explicit strategy to enable knowledge management [191] and the basis for a framework for measuring value creation [192].

Wenger summarises communities of practice as "groups of people who share a passion for something that they know how to do, and who interact regularly in order to learn how to do it better" [191]. The theory explores a number of dualities and oppositions, and the boundaries between paired concepts:

- Membership and non-membership, and their boundary, peripheral participation.
- Participation—collaborative reality-building—and reification—conversion of shared ideas into static artefacts.
- Designed and emergent behaviour, interactions, and knowledge.
- Local and global interactions and considerations.
- Identification as a practitioner and negotiation of practice.

In the context of an occupation, communities of practice provide a space outside of explicit organisational hierarchies and reporting structures where people can participate together in understanding and developing their work. A community of practice can be a place where tacit knowledge about an occupation is negotiated and communicated.

Orhun and Hopple compare theoretical frameworks for knowledge sharing in a community of practice, which are based on theories of social capital and social exchange [193]. They observe that each framework models a dimension of social capital in isolation, suggesting that the true picture may be more complex and multidimensional than any of the frameworks allows.

Nagy and Burch note specific challenges in fostering communities of practice in academia[11], as individual professional goals for academics are more likely to be self-directed or serving their discipline than their organisation; colonisation of the organisation by managerial control mechanisms leads to mistrust of organisational initiatives; and power structures are poorly defined [194].

In software engineering, Shim et al. use communities of practice as an educational tool by integrating it into their teaching, and student feedback surveys [195]. Kumar and Wallace performed a combination of observational study and participant interviews on software development team members in an organisation undergoing significant growth [196]. They used the theory of communities of practice—particularly its concept of legitimate peripheral participation—to construct a pattern language from this work that describes mentorship, process changes, personal identity shifts, and other factors that changed how the growing team performed software development.

3.6.7 Insights into RSE from the Sociology of Professions

The principal contribution to research on RSE from the sociology of professions is an understanding of how theories about occupational practice and professionalism can be developed in a field, providing useful analogy and theoretical basis for similar work in research software. Ethnographic studies of software engineers show how workplace culture and context play an important role in the adoption of professional practice, with software engineers inadvertently sabotaging their own time management practices and reading managerial politicking into the adoption of a methodology that centres empowering, self-organising interactions. Grounded theory studies of healthcare professions further explicitly link situational factors including the social and political challenges of the profession, or the availability of development opportunities, to actions taken by practitioners. These examples show that a plan to develop a theory of RSE as a profession must be able to incorporate

[11]Their research relates specifically to Australian universities, but as those institutions were initially set up as colonial exports of western-style university models and have been through a similar trajectory of increasing governmental target-setting and corporatisation to that experienced in the UK, the analogy is relevant.

broad contextual ideas, considering the research software engineers as participants in broader systems of peers, stakeholders, social norms, and other influencing factors, any of which could be important. The determination of which of those factors *are* important must come from the participants in the system and the data that they generate, and we as researchers gather.

Work on the sociology of professions also provides a useful framework for understanding what it means for an occupation to be a profession, and the pathways to professionalism. Professions are identified by a body of expert knowledge, technical autonomy, principles of service to society, and privileged status. New professions form through the identification of an occupation as a distinct "fission" from existing practice, and then must seek recognition and struggle for self-governing of their own membership and practice in competition with broader established professions or peer institutions that perceive jurisdictional conflict. These ideas provide useful context in investigating the professional status of RSE.

3.7 Conclusion

There is recognition of the dependence of modern research on software development, and of the contribution to be made to research by advancing the profession of Research Software Engineering. The development of that profession is primarily opinion-led, with those opinions unsubstantiated by evidence-based research. Researchers have separately studied the "pain points" of scientific software development, and the practices adopted by people who develop scientific software, but no work has been done to identify whether interventions including RSE ameliorate or mitigate these pain points.

To understand how to produce such a theory and to advance the RSE profession, we must look to developments in analogous fields. One very close analogy is the promotion of FAIR principles of research data management, and the occupation of data steward. FAIR shows how software systems can be considered as 'computational stakeholders' in the research ecosystem. It also provides an example of advancing practice by defining a 'manifesto' of principles that connect academic

practice both with academic values and with economic ideas of efficiency, value for money, and opportunity cost. The realisation of FAIR principles has direct implication on RSE practice, and motivates analogous development of "FAIR 4 Research Software" principles.

Another community-led development in research culture is the development of RRI, an approach to considering stakeholder needs and views which involves people impacted by the research throughout its design, implementation, and application. The RRI approach has been codified in a framework adopted by a UK research council, demonstrating that community-driven changes to research practice can become standardised.

We can also learn more broadly from the study of professions and the professionalisation of software engineering, to understand how to develop the theory needed to situate the professionalisation of RSE in its context and the current state of the art in practice. The defining 'features' of a profession are: a body of expert knowledge; technical autonomy; normative value of service to others; and high status. A theory of RSE as a profession should explain how (or if) those properties are realised and negotiated in RSE's practices and interactions.

RSE can be seen as a 'fission' of the duties of the professional researcher, but one that overlaps or parallels neighbouring fields including broader software engineering, computing, and research support or technician occupations. These overlapping occupations with their separate professional associations or organisations might lead to jurisdictional conflict; the negotiation between these fields affecting the selection and promotion of professional practice.

Research into how software engineers work provides useful guidance on how individual practice can be connected to professional and workplace values, and the context of the work. Many researchers have taken this particularly far in the healthcare occupations, using grounded theory methods to provide explaining theories describing how practitioners navigate the professional landscape from individual practice and development to advancing their professions.

Quantitative research into the effects of particular software engineering practices, such as test-driven development, typically look for effects of the practice in software or ancillary data produced by the engineers. These effects can be confounded with other factors. The researchers do not explore other motivations for adoption or promotion of the practice. This additional insight, a "theory of practice" of software engineering, could come from qualitative or mixed methods analysis, in which the reasons behind and motivations for professional practice come from the practitioners and other stakeholders themselves. Such research in software engineering is hard to find, and acknowledged by journal editors to be rare.

We address this gap in chapter 4, by designing a research methodology for a mixed-methods investigation into RSE.

4 Methodology

Contents

4.1 Introduction

As we showed in chapter 3, the field of RSE has yet to receive much attention in the scholarly literature. Quantitative studies focus on particular, narrow aspects of research software, relying on individual categories of data gathered using a single instrument, e.g. GitHub mining or surveying practitioners. We found no evidence of qualitative or mixed-methods investigations into the field, or of "big-

picture" quantitative analysis.

This absence means that leaders and practitioners in RSE lack an evidence-based view of the status of the field, and thus the ability to make testable hypotheses about its future development. Given the importance of software construction to researchers reported in section 2.4, substantive, theoretically-informed analysis of RSE could have significant impact on how research is supported and undertaken through software, and on the jobs and careers of people in the field.

We intend to contribute such an analysis of the occupation in this thesis, that can form the basis of an evidence-based "manifesto" for action in RSE in the same way that the Manifesto for Agile Software Development [42] led to substantive change in the practice of commercial software engineering.

As the author of this thesis has a long career in software engineering including working as an RSE at a university, we need to adopt an epistemology that accounts for the researcher's relationship with the field of study and incorporates reflective thinking into the analysis.

4.2 Hypotheses and Research Questions

Following the theme of a software "chasm" between research and commercial practice identified in our literature review, first proposed by Diane Kelly [107] and expanded by Tim Storer [108], we hypothesise that the chasm is real, and is perceived as a skills gap in research software. We hypothesise that the close connections between RSE and its "host" occupation of research, and the reliance on the same institutions, structures, funding sources, and dissemination routes, limits the opportunities for RSE to enact transformation of research practice, i.e. to close the skills gap. Specifically, we construct these hypotheses:

H1 Researchers working in research institutions in the UK still identify a software skills gap.

H2 Although people within the RSE community are trying to define it as a distinct profession, RSE is tightly coupled to practices and expectations in research.

H3 RSE is heavily dependent on the structures and institutions of research, and this limits the opportunity to define distinct outcomes or expectations.

H4 The more a research group "embeds" RSE into its structure, the less opportunity for distinct outcomes and expectations from the group's research software efforts.

Given the small amount of prior scholarly research in this field and lack of existing overall theory, we start with a broad intention to understand the state of RSE, and gather data to refine the direction of the research and converge on findings that confirm or refute our hypotheses. We therefore ask the following research questions:

RQ1 Do UK-based researchers have the skills and training they need to fulfil the software-related requirements of their work?

RQ2 Does RSE exhibit the four properties of a profession in the functionalist theory of professions: a body of expert knowledge; technical autonomy; valuing service to others; and privileged status?

RQ3 How do the institutions and logics of research influence the way an RSE performs their work, and what competing institutions vie for legitimacy?

RQ4 Does the structure of the relationship between researchers and RSEs in a research group affect the research software practice?

RQ1 addresses H1 by measuring the presence or absence of a software skills gap in research. RQ2 addresses H2 by uncovering whether RSE has a distinct professional identity that can be separated from the researcher identity. RQ3 addresses H3 by discovering whether RSE has its own ways of working or professional controls and influences that lead to practices that would not be found in non-RSE research activities. RQ4 addresses H4 by investigating whether different ways of organising researchers and RSEs are associated with distinct outcomes in the production of research software: in other words, the effect of group structure on the software skills gap.

4.3 Applicable Theories of Professions

While RSE has yet to receive systematic study, we reviewed the literature on the study of professions—including software engineering—in chapter 3, which provides useful theoretical bases and examples of instruments that we could apply to our own research. Because there is limited existing literature in the field it is difficult to claim that any one theoretical approach has clear benefits or drawbacks when applied to RSE. Therefore, we design a broad methodology, in which we apply different theoretical analyses to each component of our research.

Early research into the organisation and work of professions used a functionalist approach, identifying the properties that distinguish professions from non-professional occupations. This approach assumes a distinction between professions and other occupations, particularly self-curation of the profession's membership and body of knowledge, and a paternalistic attitude towards application of the profession's knowledge in society. While making such assumptions in relation to modern professions—in which professionals are employed and appraised directly by interested organisations rather than maintaining a client-agent relationship—is problematic, the attributes identified by functionalist theories offer interesting discussion points to start conversations with practitioners about how they view their field. We follow this approach in chapter 6, to frame discussions with participants and learn how they perceive the RSE occupation.

Among the theoretical variety observed, some approaches—notably the Marxian and Weberian theories—make particular assumptions about the relationship between workers and the state. These theories seem to "beg the question" by embedding assumptions about those relationships into the hypotheses generated and results drawn, and as a result we choose not to adopt those approaches in our work. We instead look for discourse-based theories that foreground the perspectives of the participants, and draw information about their work and their relationships from those perspectives.

Neo-institutional theory (NIT) interprets an individual professional's actions as influenced by multiple institutions. The institutions may be formal organisations

such as the individual's employer, a professional association, or the state, or informal associations like ad-hoc communities. Each institution proposes "logics" which justify courses of action and, given an individual's situation, they weigh up the relative legitimacy of the institutions and select a logic to follow. The institutions therefore engage both directly in defining the logics that describe how people (should) do their work, and in legitimising activities that promote their own legitimacy and demote those of competing institutions. We use NIT in chapter 7 to investigate the professional controls and influences on RSE, through the lens of blog posts on the topic (see section 4.6.4). Blog-writing itself can be perceived as a legitimising activity, as the authors justify their own approaches to work and the institutions that support those approaches, so NIT provides a helpful link between this "evangelism" activity and professional practice.

The theory of communities of practice places more emphasis on the ad hoc and informal environments that practitioners create when they "learn together" by sharing experiences and building an understanding of their work. While not limited to the workplace, communities of practice has frequently been applied to studies of occupations and professions due to its explanation of how professional knowledge doesn't necessarily flow along formal channels, and its highlighting of the building and sharing of tacit knowledge in an occupation. The transference of tacit knowledge is important to the software craftsmanship model of the software engineering profession, discussed in section 2.3.4.

We use communities of practice in chapter 8 to explore how RSE is embedded in research practice. Research groups engaged in writing research software "learn together" on solving their research problems and developing the software to address those problems, and the communities of practice theory helps us to understand how the relationships between the people in the groups and their organisation of their work supports that communal learning.

We compare the analyses and conclusions drawn in each of the research chapters in chapter 9, to cross-check results, discover contradictions, and to build a more holistic view of RSE.

4.4 Constructivist Approach

To be clear about the approach to analysis in our research and the derivation of our results, we need to explicitly state our research epistemology. We adopt a constructivist, rather than a positivist or post-positivist approach, because professional working is a socially-constructed activity that undergoes continual development and negotiation as people seek to satisfy the needs of others or advocate for their own approaches and desires. We do not believe there to be an underlying objective "reality" of research software engineering in the same way that one might accept a basic truth of gravitation, and a positivist approach could introduce the researcher's bias in looking to identify "essential" traits of the field, and overlook the contributions that the researcher's relationship with their participants can make to the data collected. In particular, the author of this thesis has worked as an RSE, so participants may discuss their work with a peer or power relationship in mind.

We initially set out to derive a grounded theory (GT), but found challenges in recruiting sufficient participants to achieve theoretical saturation—the situation in which new data contains no new codes and categorisation is fully developed [164, p.96]—and realised we would not be able to complete such an approach within the necessarily limited timespan of a doctoral programme of study. We therefore designed a mixed-methods programme of study that represents a snapshot of RSE professional practice and presents analysis informed by the epistemological foundations and methodology of constructivist GT, that forms the preliminary basis of a grounded theory of the occupation. In chapter 10 we identify further work that could develop the results we present into a GT.

4.5 Responsible Research

Taking a responsible research approach to this research itself by sharing interim results with RSEs and participants, and eliciting feedback on the direction and application of the research, will increase the sense of engagement that RSEs have with the research and recognition of their own interests and concerns in its development.

This acts as an important counter to the prevailing attitude that much research in software engineering is not perceived as relevant in industry due to a lack of engagement with practitioners [197], and addresses the call by Goth et al. for RSEs to be guided by ethical values [88, §3].

4.6 Research Methods

As noted in section 4.2, our research questions are designed to initially get a broad sense of the state of RSE, and to explore interesting aspects of the field that develop from the data.

4.6.1 Mixed Methods Approach

In section 4.4, above, we noted that the initial goal of this project was to create a GT that explains the state of the RSE occupation, and that in early phases of the work we identified that we had not been able to achieve theoretical saturation. We therefore altered our approach and designed a mixed methods study, which uses both qualitative and quantitative instruments to explore its research questions.

Creswell and Plano Clark identify a typology of distinct mixed methods designs [198]; this work follows a sequential, exploratory design, in which qualitative methods are dominant. We present the four phases of the research in chapters 5—8 in the order in which they address our numbered research questions, and present the comparison between these instruments in chapter 9. Here, we give a brief timeline of the phases, to explain the development of the work and the selection of instrument at each phase.

1. The first phase was a round of open-ended interviews (qualitative), designed to explore the research area from the participants' perspective; see section 4.6.3. We were unable to achieve theoretical saturation during this phase due to a slow pace of recruiting participants, and the clustering of respondents in scientific fields.

2. The interview data revealed interesting power relationships between RSEs and researchers, and we were interested to find out how these relationships manifest in the public performance of RSE, as distinct from the relatively private nature of a one-on-one interview. We therefore turned to blog data (qualitative) for the second phase of our study; see section 4.6.4. Discovering that the blogs didn't address all of the gaps in the interview data, we identified in this phase that we were not on track to achieve theoretical saturation to develop a GT and re-designed our research programme as a sequential, exploratory mixed methods study.

3. Analysis of the blogs indicated potential skills and training gaps in RSE, leading us to ask whether the gaps in software engineering capability identified in the literature were being addressed. We designed a survey (quantitative) to explore the nature of the capability gap; see section 4.6.2.

4. Finally, the survey results showed a lack of RSEs mentoring researchers in software skills, which raised the question: how do RSEs and researchers work when they work together? We designed focus groups (qualitative) to bring whole research groups together and understand how they collaborate; see section 4.6.5.

We triangulated findings from these instruments as the research progressed to develop our theoretical understanding of the field. The triangulation is presented in chapter 9.

4.6.2 Survey

We investigate RQ1 by implementing a survey to capture an up-to-date quantitative view of the "software chasm" and the software skills gap experienced in research in the UK. This survey is designed to investigate the confidence researchers have in their software engineering abilities, and the routes to knowledge and skills acquisition available to them.

We distributed the survey among researchers and students in the University of Oxford, and received 60 responses.

Full details of our method and results are presented in chapter 5.

4.6.3 Interviews

We then undertake a round of open-ended interviews, exploring the properties of professions identified in RQ2. The interview participants include RSEs, RSE group leaders, and other people who work alongside RSEs.

When adopting a constructivist epistemology, the relationship of the interviewer and interviewee is an important consideration in the design of the interviews and analysis of interview data [165, p.90-94]. Given the author's prior experience as an RSE and participation in online discussion forums and in-person conferences, we expect participants to be open to discussing the field in depth, for any technical argot used to be understandable, and to be readily able to guide conversations toward our sensitising topic of the properties of professions listed in RQ2. We use memoing as a reflexive technique to consider our own position and experiences with respect to the interview data, and to check that coding and derived categories come from the data rather than from our preconceived model of the field.

We conducted the interviews using Microsoft Teams. To preserve situational details, in addition to taking field notes during the interviews, we recorded the video feed of the interviews and transcribed the conversations, before anonymising the transcripts. We referred to all three media during coding, which we performed in three stages:

1. Initial coding, in which we coded each sentence in the interviews for ideas, events, and feelings.

2. Axial coding, in which we identified groups and abstractions in the initial codes.

3. Theoretical coding, in which we connected the groups identified in axial coding into higher-order categorisations and linked categories with the properties of professions identified in RQ2.

Full details of our method and our results are presented in chapter 6. The recruitment messages we send to potential participants, and the interview script we used, are included in appendix A.

4.6.4 Analysing RSE Blogs

To explore the institutions and logics used by RSEs to navigate their occupation and choose how to perform their work, the subject of RQ3, we investigate public blog posts written by and for RSEs. Blog articles are an appropriate subject to explore this question as they represent "performative" work; the authors define and justify the work they do, and advocate for the legitimacy of their approach, in the articles, and share them publicly to socialise and normalise those justifications.

We designed search terms to discover blogs by and for RSEs, and used the search engine DuckDuckGo to find candidate websites. We then filtered our results to websites that have RSS feeds indicating chronological sequences of blog posts, and found posts published within the date range January–April 2023 to form a contemporary snapshot of RSE performance.

Our analysis uses a similar series of coding rounds to the interview data. We apply the neo-institutional theory of professions (NIT), categorising our sentence-level initial coding based on the institutions and logics present in the blog posts. Additionally, we connect the codes to the same categories identified in the interview data, to permit comparison between the interviews and blogs.

Full details of our method and our results are presented in chapter 7.

4.6.5 Research Software Focus Groups

In our final phase of primary research, we investigate the communities of practice that form when researchers and RSEs work together by hosting focus group sessions that see members of a research group completing activities together that investigate how

they perform their work. We use this technique to explore how the organisational relationship between researchers and RSEs in a research group relates to the software engineering practices undertaken in that group, the subject of RQ4.

We organised focus groups that each comprised members of a single research group, that were held in-person in a workplace setting. We prompted the participants in each focus group to complete two activities: in the first they constructed a diagrammatic representation on a whiteboard of their work and how each of the members of the research group is involved in that work; in the second they discussed the relative importance of properties of a software engineering endeavour to their research group, in addition to their group's capability with respect to each aspect.

The focus groups were filmed from multiple angles, and we captured images of the whiteboard they used in the first activity, and the sets of cards they used in the second activity, in addition to taking field notes.

We transcribed the video recordings, and analysed the data by applying the theory of communities of practice. Finally, we coded the transcripts in NVivo to permit comparison with interview and blog data.

Full details of our method and our initial results are presented in chapter 8.

4.7 Ethical Approval

The Departmental Research Ethics Committee in the Department of Computer Science, University of Oxford reviewed and approved our work, reference CS_C1A_021_043 (for the interviews) and CS_C1A_24_001 (for the survey and focus groups).

4.8 Conclusion

We present a mixed methods programme of research to investigate the field of research software engineering in the UK. Our programme draws from multiple theoretical backgrounds—the functionalist theory of professions, neo-institutional

theory, and communities of practice—to analyse our findings. This approach is appropriate because RSE is relatively unexplored by existing scholarly work, and there is therefore no preferred or favourable theoretical basis for understanding the field.

This work draws on our literature review—presented in chapter 3—and represents a novel contribution to the understanding of RSE and, more generally, to the understanding of academic software development practice.

We undertake four distinct components in our mixed methods programme. The quantitative component is a survey of software engineering experience among researchers and students. The qualitative components are: a series of interviews with people who work in research software; a search and analysis of blog posts about research software engineering; and a preliminary collection of focus groups comprising researchers and RSEs who work together on producing research software.

We present this work in chapters 5-8. We compare the findings of each component of the work in chapter 9, and draw conclusions and outline future directions for the work in chapter 10.

5

Measuring the Chasm—Surveying the Research Software Skills Gap

Contents

5.1 Introduction

To investigate RQ1, whether researchers working in UK research institutions still recognise the skills gap reported by Storer in [108], we designed a survey that asks researchers about their experience using and developing research software, and how they acquire their software skills. As seen in sections 3.2.4 and 3.5.1, surveys have been applied by researchers to scientific computing and to software engineering, and we expect the method to apply to RSE well.

We sent the survey to all researchers in Oxford University, via divisional mailing lists, the Oxford Research Software Developer Network, to the Mathematical,

Physical, and Life Sciences (MPLS) Doctoral Training Centre, and to the Digital Scholarship @ Oxford (DiSc) group in the Humanities division and Bodleian Libraries. The choice to distribute the survey within Oxford University was a pragmatic one made to balance the convenient access to a broad researcher population with the bias inherent in receiving responses from a single institution. Our alternative routes to reach a wider audience were mostly biased towards RSEs and other interested people—who are more likely to be skilled at developing research software than the average researcher. As de Mello and Travassos argue, we cannot know the representativeness of a sample without a census of the population, so aiming for a *heterogenous* sample is a sufficient proxy [199].

We received 60 responses.

5.2 Method

We followed the seven-stage survey method timeline described by Molléri et al. [200] Their timeline is based on a comprehensive review of survey guidelines in the software engineering literature, and represents recommended practice among software engineering researchers. The seven-stage approach appears in both [201] and [202].

Identify Research Objectives

Our objectives are to contextualise the role of RSE in contemporary research, by identifying the distribution of software engineering skills among researchers, and the need for support in developing research software. Pursuing these objects provides an answer to RQ1: the extent to which researchers need support in developing research software is a measure of the "software skills gap" among researchers.

Identify Target Audience

To answer RQ1, the target audience for the survey needs to be researchers in the United Kingdom. For ethical reasons, we further restrict our audience to adults

(over 18 years old, in the UK); in practice we expect this to exclude a very small number of doctoral students and few other people.

Design Sampling Paradigm

Motivated by the pragmatic constraint of completing the survey within the timescale of our doctoral research programme, we elect to use a convenience sample of researchers at Oxford University. We believe that this is a large and diverse cohort that covers much of the population in the target audience, even though surveying researchers at a single institution introduces systematic bias effects. Through one of our supervisors (Prof. Gavaghan) we shared the survey with a mailing list that includes all researchers in the university[1], and, via divisional administrators, to mailing lists that are circulated to researchers in each of the university's divisions, in addition to a doctoral training centre, ensuring good coverage within our convenience sample. We additionally sent the survey directly to a digital humanities group, to attract responses from outside of scientific disciplines.

Design Survey Instrument

We followed guidelines from *Asking Questions: A Practical Guide to Questionnaire Design* [203]. Sudman and Bradburn advise starting the survey with simple questions that have multiple choice answers. Respondents quickly get into the "swing" of answering these questions, and have mentally committed to responding to the survey before they reach any taxing or challenging questions that increase the cost of commitment.

By "simple" questions, the authors do not merely mean those which are easy to understand and require short—preferably categorical—answers. They also mean questions that do not challenge the respondent's identity or the image they would like to project of themselves. Not only should questions about behaviour seen as potentially antisocial or aberrant be asked later in the questionnaire, but also demographic questions as they could threaten the respondent's understanding of

[1] Any researchers who have explicitly opted out by unsubscribing from the list are not included.

their identity. The only reason to ask such questions early in the survey is when they are used to filter, or conditionally include, other questions.

Sudman and Bradburn also recommend asking funneling questions—those which direct respondents down particular paths of follow-up questioning—and filtering questions—those which exclude certain follow-up questions—early in the survey. One reason to do so is to avoid asking questions that later turn out to be irrelevant by ensuring that the filtering questions are asked before the questions contingent on the filter. Their main reason, however, is that asking the filtering questions immediately before the filtered questions allows respondents to learn which route will yield the shortest path through the survey, and tailor their answers to that rather than give an accurate account.

This problem is clearest to understand in a paper-based self-administered questionnaire such as a mail-in form. If a page contains the question "12. Have you ever smoked?" followed by the statement "If you answered 'no' to question 12, skip to question 17" then the respondent can quickly read ahead to discover that they'll have a shorter route if they answer "No" to question 12, whether or not they smoke. If all of the filtering questions are on the first page, then the respondent may answer them before discovering later in the questionnaire that they affect the path through the rest of the questions.

The effect is not limited to self-administered forms, though. A respondent may learn during a telephone or online survey that questions of the form "Have you ever. . . ?" are followed by a page of details about that activity, and elect to respond "No" whenever that form of question is asked. This can be mitigated in the same way as with a paper form, by asking the filtering questions at the beginning of the survey and coming back to the detailed questions later.

This idea of people "cheating" themselves out of longer paths through the survey indicates that longer surveys put respondents off participation. Surveys should be as short and simple to complete as possible—though not to the point where important information about the question being researched is lost.

Survey results are known to be subject to acquiescence bias, in which the distribution of responses to a Likert-type question[2] skews systematically more toward agreement with the statement than would be expected if respondents were answering truthfully. Researchers can account for this bias by reversing the sense of different questions in the survey, so that the bias toward agreement would be equally weighted in favour of and in opposition to the effect being measured.

A similar bias is exhibited toward desirable characteristics, beliefs, or personality traits. For example, respondents to a questionnaire on using a mobile phone while driving might systematically under-report their tendency to do so as the behaviour is considered socially unacceptable. Survey designers can try to get respondents to focus on facts, for example how long it has been since the last time they used their mobile phone whilst driving, to reduce the challenge to the respondents' self-perception.

Another technique discussed by Sudman and Bradburn in [203] to improve responses to "challenging" questions is to normalise the behaviour deemed undesirable by citing how often it's done, or how many people do it, or producing a justification for the behaviour. Care needs to be taken over this approach to ensure that it doesn't lead respondents into biased answers by making the previously-embarrassing behaviour seem rational or even expected.

Sudman and Bradburn report that researchers disagree on whether Likert-type rating scales should include or exclude the neutral middle option. When it is included, the scale has an odd number of responses, for example "strongly disagree [with the statement in the question]", "disagree", "neither agree nor disagree", "agree", "strongly agree". When it is excluded, the scale has an even number of responses, and there is no option for "neither agree nor disagree".

Proponents of the included middle argue that it allows respondents to truthfully indicate when they have a neutral opinion on the question topic, which is lost if the neutral response is not available. Opponents of the option argue that it allows

[2] "To what extent do you agree with the following statement:..."

respondents to acquiesce and fail to give a position on a topic that, were they to reflect on, they could find even a weak argument in favour or against.

Taking these factors into account, we designed the survey instrument below. We based the survey on an existing example created by Dr. Martin Robinson to create a "business case" for a research software engineering group at Oxford University, which he shared in personal communication.

1. This survey is intended to gather information on the importance of research software to researchers' work, and is approved by the Computer Science departmental ethics committee, reference CS_C1A_24_001. By checking this box, you agree that you are over 18, consent to the use of your responses in [the author]'s doctoral thesis, and to avoid entering personally identifiable information in any response. For information about the research or to withdraw before 31st October 2024, email [the author].

   The respondent may either indicate agreement, or withdraw from the survey.

2. Which division are you based in?[3]

   - Humanities
   - Mathematical, Physical, and Life Sciences
   - Medical Sciences
   - Social Sciences
   - Department for Continuing Education
   - Gardens, Libraries, and Museums

3. Which of these roles best describes you?

   - Student
   - Postdoc
   - Lecturer

[3]In the University of Oxford, a division is an administrative unit that comprises departments in related subject areas.

- Group leader
- Professor
- Reader
- Research Software Engineer
- Other [The respondent can enter a free-text answer.]

4. Do you, or your group, use or develop research software? Research software is "Software that is used to generate, process or analyse results that you intend to appear in a publication (either in a journal, conference paper, monograph, book or thesis). Research software can be anything from a few lines of code written by yourself, to a professionally developed software package. Software that does not generate, process or analyse results - such as word processing software, or the use of a web search - does not count as 'research software' for the purposes of this survey." [source: https://www.software.ac.uk/blog/its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers]

   - Yes
   - No [The respondent skips to question 10, though this is not presented in advance to avoid respondents choosing "No" as the shorter route through the survey.]

5. Which of the following categories best describes the research software you or your group uses or develops? Choose as many as are relevant.

   - Data processing
   - Modelling and simulation
   - Web development
   - Embedded development
   - Mobile apps
   - Machine vision or image processing

- Equipment control
- Other [The respondent can enter a free-text answer.]

6. Roughly what percentage of your time do you spend directly using or developing research software? Don't include time you spend managing other people who use or develop research software.

   - 0
   - 20
   - 40
   - 60
   - 80
   - 100

7. How important is research software to your work? [The respondent chooses a point on a five-point Likert scale, with point 1 labeled "Not at all important" and point 5 labeled "Vital".]

8. Does anyone in your group or lab develop their own research software?

   - Yes, me or others including me. [Question 9 is only shown if the respondent chooses this option.]
   - Yes, others do but not me.
   - No.

9. Which of the following best describes how you have obtained the skills you have needed? You may choose multiple options.

   - Self-taught
   - Mentoring from a research supervisor or experienced researcher
   - Mentoring from a research software engineer

- Formal training in programming or software development in the University
- Formal training in programming or software development outside the University
- Formal training in software engineering in the University
- Formal training in software engineering outside the University

[The following questions are asked of all respondents, independent of their answers to earlier questions.]

10. Have you or your group/lab ever hired someone specifically to develop research software?
    - Yes [Question 11 is only shown if the respondent chooses this option.]
    - No
11. Who did you hire to develop research software?
    - A co-investigator who took responsibility for the research software
    - A research assistant tasked with writing software
    - A research software engineer
    - A software technician
    - An external consultant or contractor
    - Other [The respondent can enter a free-text answer.]
12. With your current resources, could you recruit someone full-time specifically to develop research software?
    - Yes
    - No
13. What would be your single preferred way to work with someone developing research software?

- Train all researchers in my group to develop research software
- Some researchers, or one researcher, in my group focuses on research software
- My group would include one or more research software engineers
- My group would use the services of a pooled Research Software Engineering group in the University
- My group would use the services of external contractor or consultancy
- Other [free entry]

Validate the Survey Instrument

We input the question into the JISC online surveys platform, and shared it with our supervisors to validate the instrument. The logic and wording above shows the state of the survey after validation, with the simplification that free-form responses to multiple choice questions are presented inline, whereas JISC online surveys requires two questions, with the free-form response being shown only if the respondent chooses "other" as their answer for the multiple choice question. For example, question 3 above, "Which of these roles best describes you?" is presented as one question above, but is two questions (questions 4 and 5) in JISC online surveys. The follow-on question is only shown to people who choose "Other" for the first question, and is titled "Please describe your role."

In the rest of this chapter, we refer to questions by their number as presented in 5.2, and do not use the number assigned by JISC online surveys.

Distribute the Questionnaire

We crafted the following recruitment message:

> Dear researcher,
>
> My name's Graham Lee, and I'm studying how researchers use and create research software for my D.Phil in the Department of Computer Science, at the University of Oxford, supervised

by Professor David Gavaghan. As part of my research I've created an online survey that asks about your experiences with research software, and I'd like to invite you to take part.

If you agree, and are over 18 years old, please complete the survey at [URL], which will take approximately 20 minutes. Your responses will be anonymous, and I will analyse the reponses in aggregate and include the results in my doctoral thesis, to be submitted in January 2025. You will be able to withdraw your participation at any time before 31st October 2024. More information, including my research question and the research ethics information related to this study, is available in the attached information sheet. Please contact me by email graham.lee@cs.ox.ac.uk if you have any questions.

Kind regards,

Graham.

We distributed the recruitment message and survey link to divisional administrators via the Oxford University Research Staff Hub, and additionally posted the message to a Slack group operated by the Oxford Research Developer Network. We received 60 responses between 1st March 2024 and 1st August 2024, and no requests from participants to withdraw from the study.

Analyse and Report Results

All 60 respondents confirmed that they were over 18 years old and agreed to participate (question 1).

The distribution of responses to question 2 "Which division are you based in?" is shown in figure 5.1. None of the participants reported being in the Department for Continuing Education or the Gardens, Libraries, and Museums divisions.

As shown in figure 5.2, the majority of respondents (54) were students. One person wrote another option ("Former DPhil Student"), and no participant reported being a Lecturer or Reader.

Only one respondent—a student in the Mathematical, Physical, and Life Sciences division—reported not using or developing research software in their group's work,

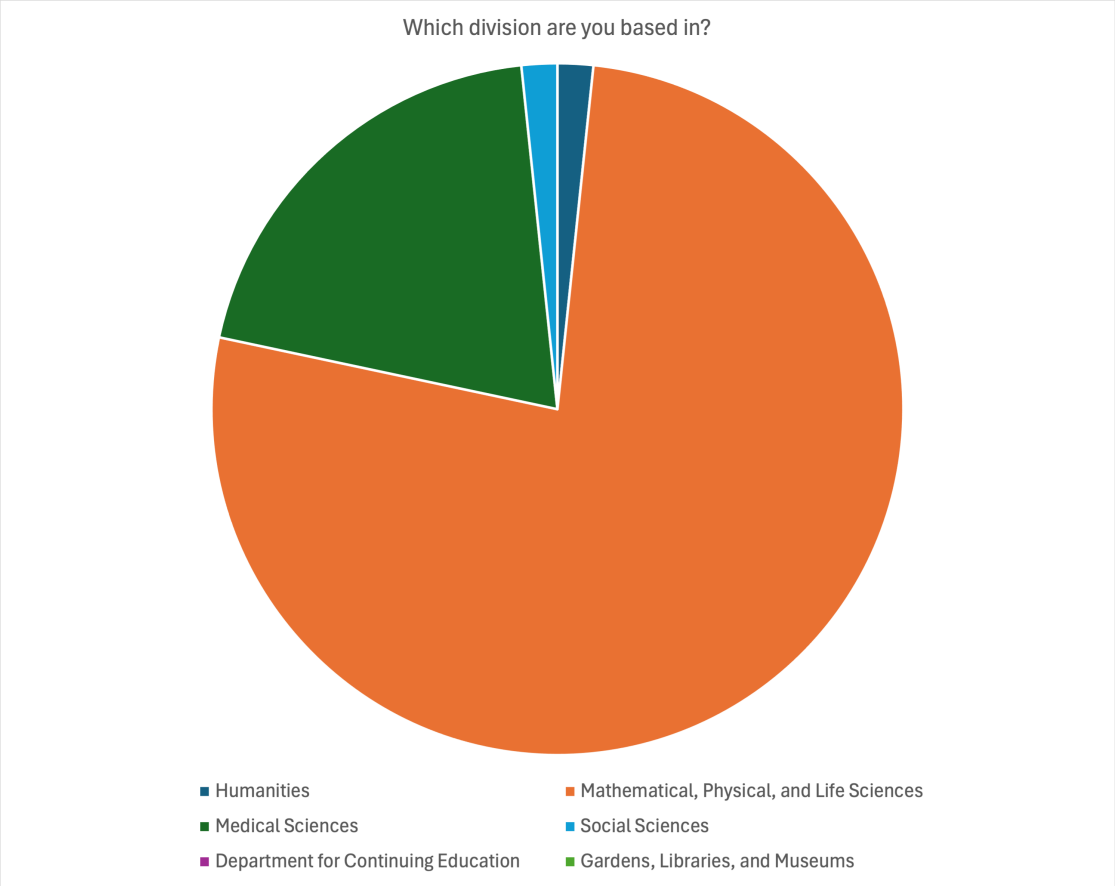


Figure 5.1: Responses to the question "Which division are you based in?"

so questions 5-9 each had 59 responses. Question 5 asks "Which of the following categories best describes the research software you or your group uses or develops?", and the responses are summarised in figure 5.3. Because respondents can choose more than one category, the total number of answers to this question is greater than the number of responses.

No respondents chose "Embedded development", and one respondent gave another option ("Bioinformatics (mainly processing genomic data) and statistical analysis of these processed datasets")—this respondent reported being a group leader in the Medical Sciences division. The single response for "Mobile app" was from the one respondent who reported being in the Humanities division.

Interpreting these results through the lens of Nieuwpoort and Katz's roles for scientific software [6], we find that a vast majority of the software developed by respondent's groups relates to components of physical or virtual instruments (the

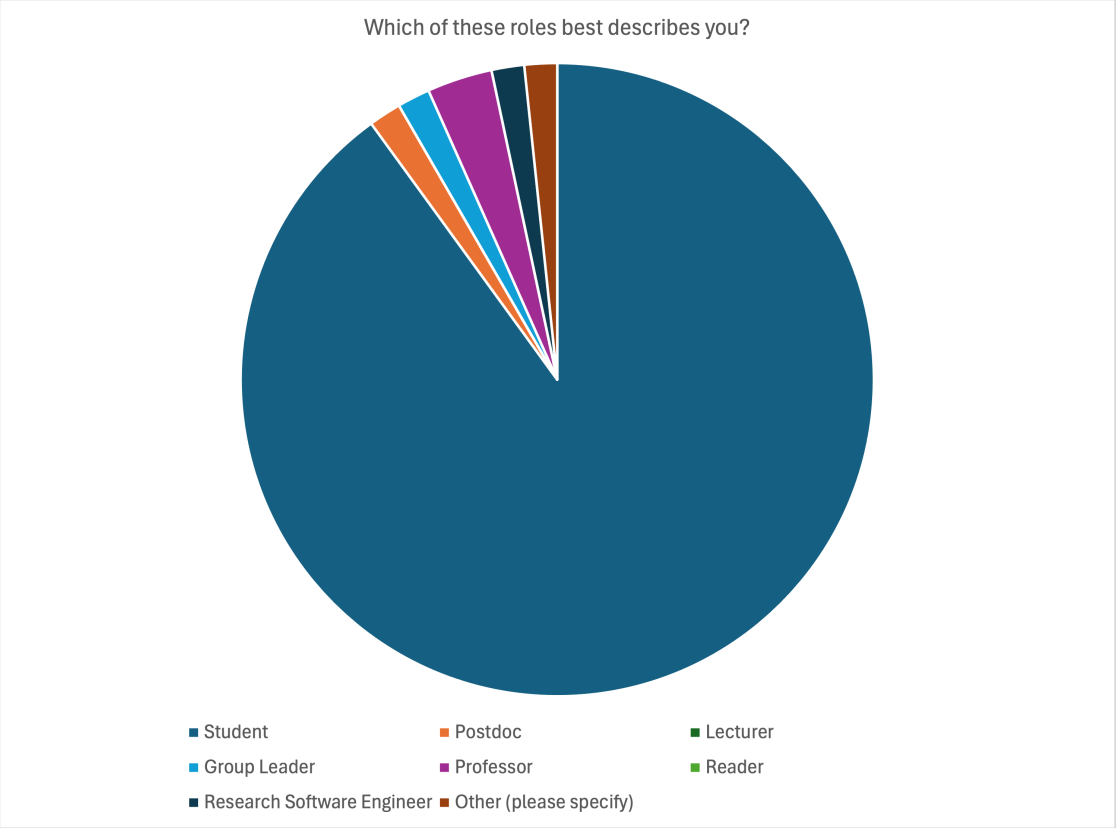


Figure 5.2: Responses to the question "Which of these roles best describes you?"

"data processing" (53), "machine vision or image processing" (20), "equipment control" (8), and one "other" choices, totalling 82 of 132 answers (62%). "Modelling and simulation" (45 answers, 34%) matches their "an instrument itself" category, so in total 127 of 132 instances (96%) of software developed by research groups reported in this survey represent instrumentation in the Nieuwpoort and Katz typology.

The remaining options are ambiguous in relation to Nieuwpoort and Katz's model; "web development" and "mobile apps" refer to the deployment mechanism for the research software, not its role in the research lifecycle.

The distribution of responses to question 6, "Roughly what percentage of your time do you spend directly using or developing research software?", and question 7, "How important is research software to your work?", are shown in figures 5.4 and 5.5. No respondent reported that research software is not at all important, or not very important, to their work; in other words 100% of respondents consider software to be at least somewhat important to their work.

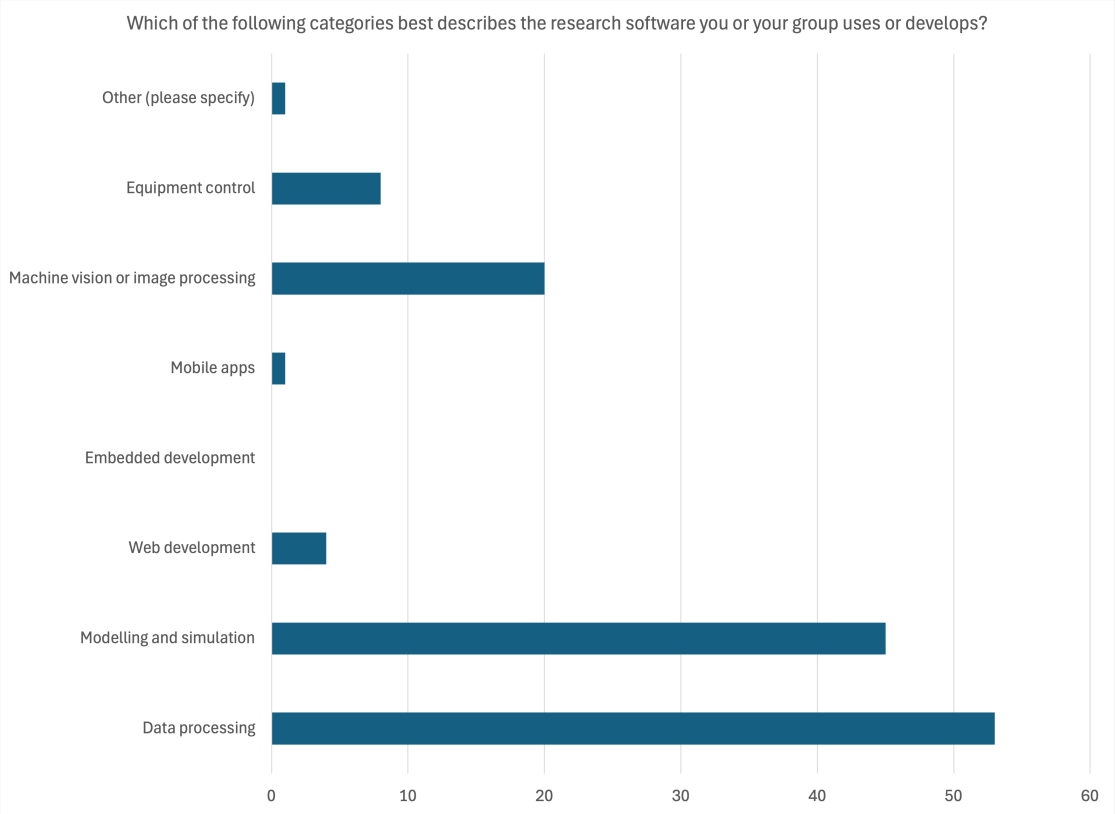


Figure 5.3: Responses to the question "Which of the following categories best describes the research software you or your group uses or develops?"

The responses to question 8, "Does anyone in your group or lab develop their own research software?", are summarised in figure 5.6.

The nine respondents who replied "No"—indicating that nobody in their research group develops research software—are all students or former students: five in mathematics, physical, and life sciences; three in medical sciences; and one in humanities. Both of the professors who responded (both in mathematical, physical, and life sciences) are among the 13 people who answered that other people in their group, but not themselves, write research software, along with a student in social sciences, three students in medical sciences, and seven students in mathematical, physical, and life sciences.

Of the 37 people who answered "Yes, me, or some people including me", one is an RSE, one a group leader, one a post-doctoral researcher, and the remainder are all students. Six—including the postdoc and group leader—are in medical

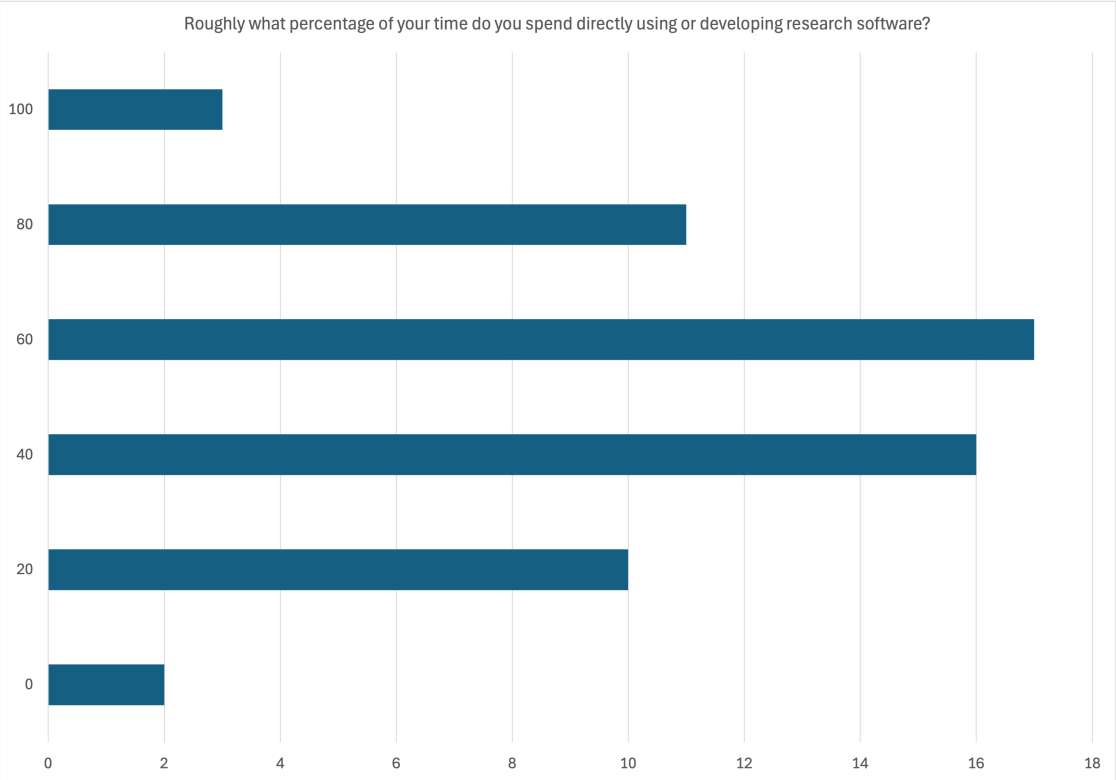


Figure 5.4: Responses to the question "Roughly what percentage of your time do you spend directly using or developing research software?"

sciences, and 31 are students.

Because 37 people answered "Yes, me, or some people including me", there are 37 responses to question 9, "Which of the following best describes how you have obtained the skills you have needed?", summarised in figure 5.7. Because respondents could choose multiple responses to this question, the total number of answers to this question is greater than the number of responses.

The following question was available to all respondents to answer, regardless of their pathway through the earlier part of the survey. 21 respondents (35%) answered "Yes" to question 10, "Have you, or you group or lab, ever hired someone specifically to develop research software?". All remaining respondents (39 individuals, 65%) answered "No". The 21 respondents who answered "Yes" were asked question 11, "Who did you hire to develop research software?", and their answers are shown in figure 5.8.

Because respondents could choose multiple responses to this question, the total

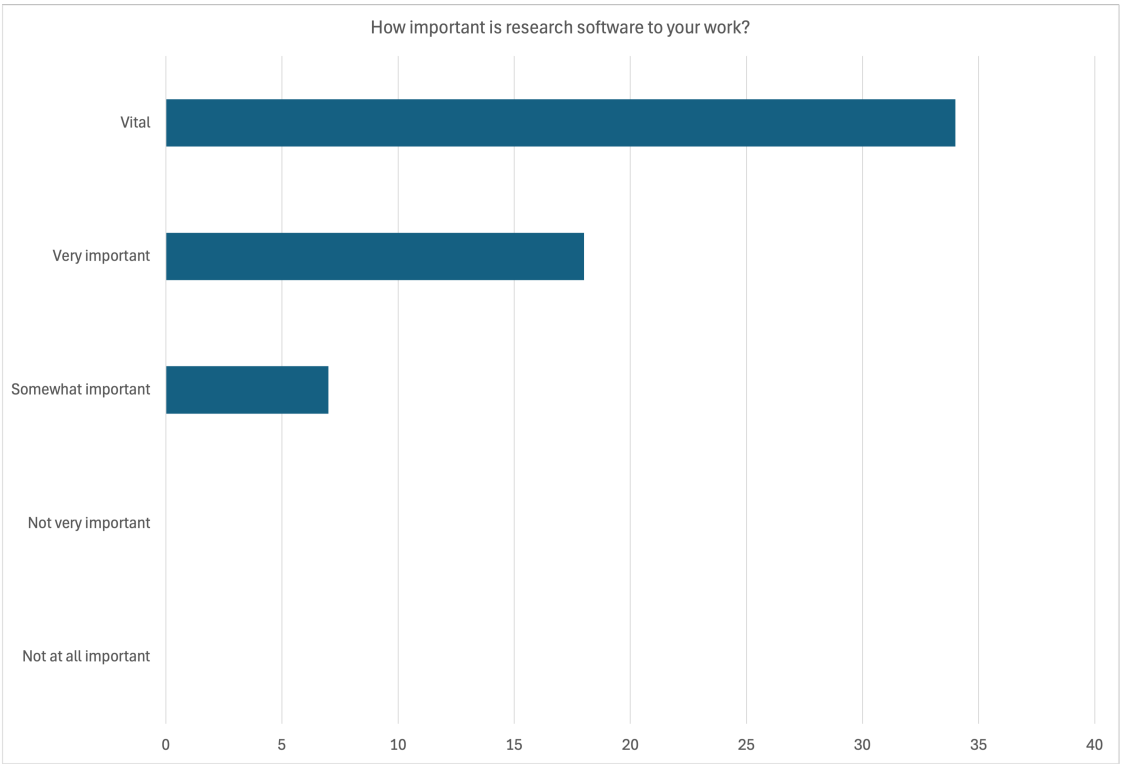


Figure 5.5: Responses to the question "How important is research software to your work?"

number of answers is greater than the number of responses. The two "Other" responses are "Post-doc", and "I didn't hire them but in my supervisor's group meetings, I think some of the other people in the group meetings are research software engineers, but I don't really work with them directly. They do sometimes give general advice in meetings though."

As noted in section 5.2, longer surveys can discourage participation from respondents. The remaining questions in our survey did not achieve 100% completion, probably due to their position at the end of the survey. 58 respondents (97%) answered question 12, "With your current resources, could you recruit someone full-time specifically to develop research software?", with 12 (20% of all respondents) answering "Yes" and 46 (77%) answering "No".

54 respondents (90%) answered question 13, "What would be your most-preferred way to work with someone developing research software?", and the responses are summarised in figure 5.9. Two "Other" responses were recorded, one was "N/A"

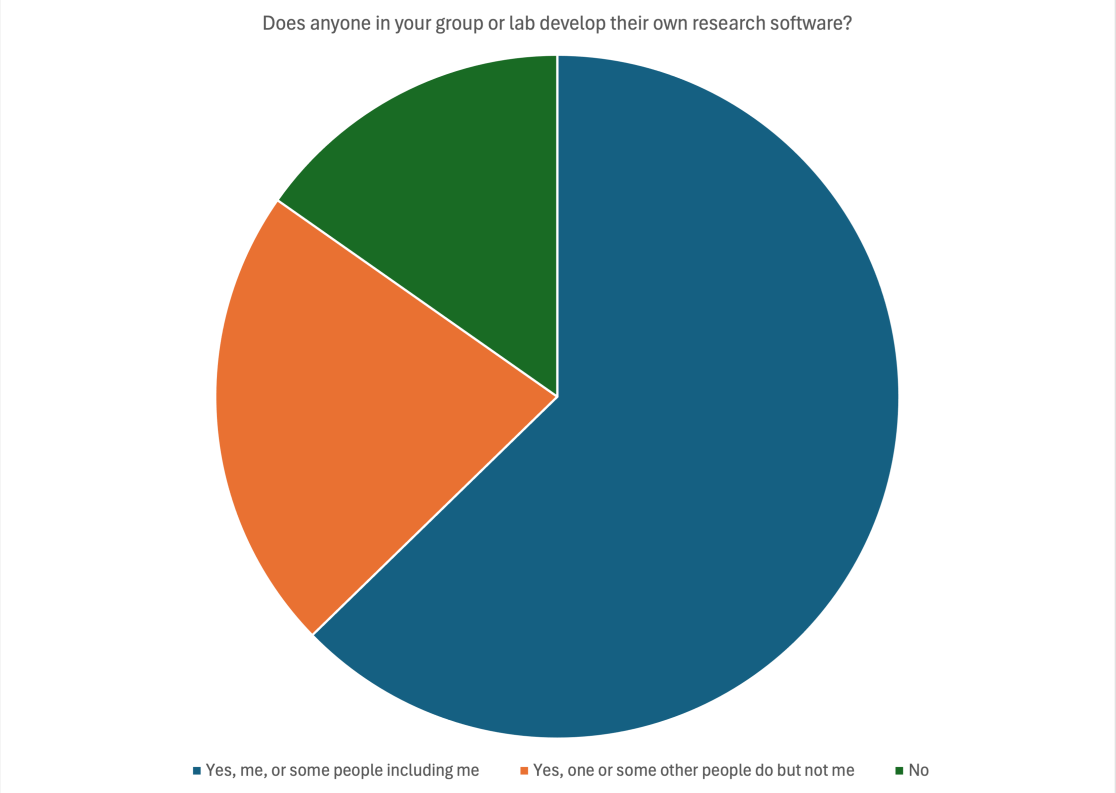


Figure 5.6: Responses to the question "Does anyone in your group or lab develop their own research software?"

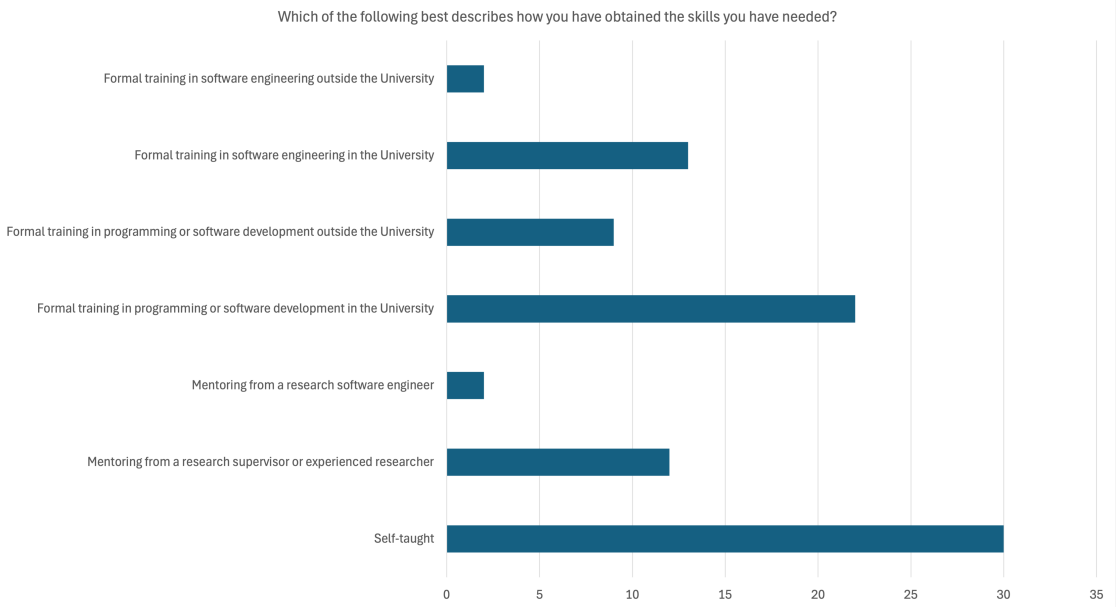


Figure 5.7: Responses to the question "Which of the following best describes how you have obtained the skills you have needed?"

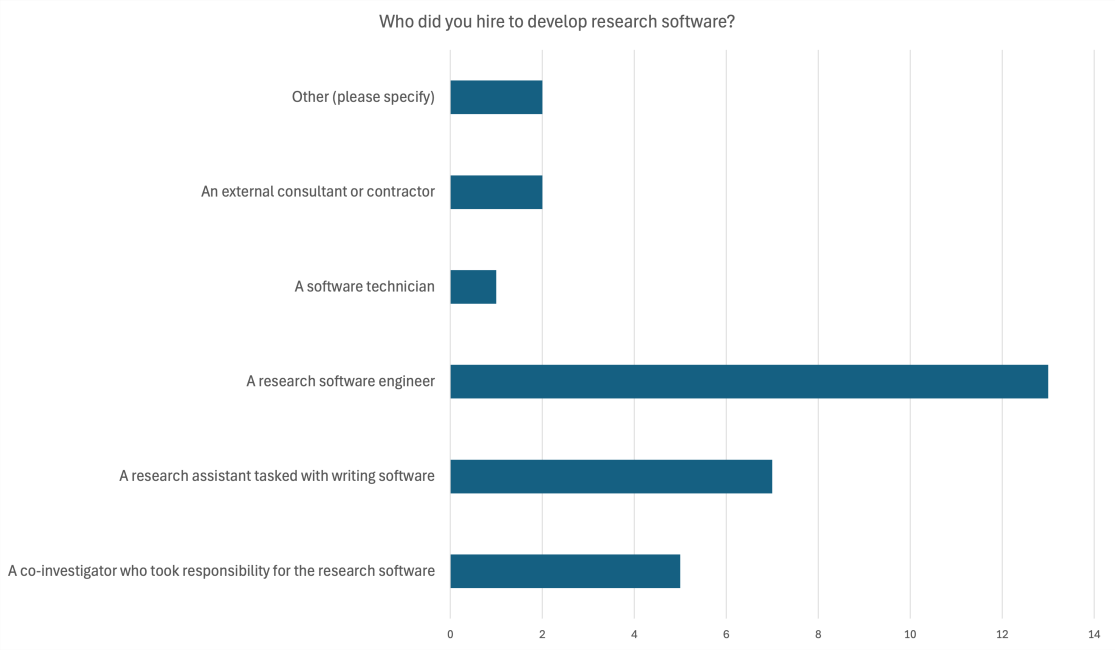


Figure 5.8: Responses to the question "Who did you hire to develop research software?"

which we considered an absent response, so that the number of true responses is 53 (88%). The remaining "Other" response is "I'm just a student but I think it would be best to train everyone - so the answer 'We would train all researchers in my group to develop research software"', which we counted as a response for that answer.

5.3 Discussion

The University comprises more than 26,000 students and 16,000 staff [204], so our sample of 60 respondents represents 0.14% of the population. Our sampling strategy did not explicitly target students—one of our supervisors (Prof. Gavaghan) circulated the survey recruitment message on a doctoral training centre mailing list after other approaches yielded few respondents—so a low response rate among students is not surprising. In fact the doctoral training centre recruited around 12-18 students per year up to 2023 [205], so 54 student responses could represent full turnout: if a professor sends out an e-mail asking people to respond to a survey, students are more likely to feel pressure to comply due to the power dynamic of

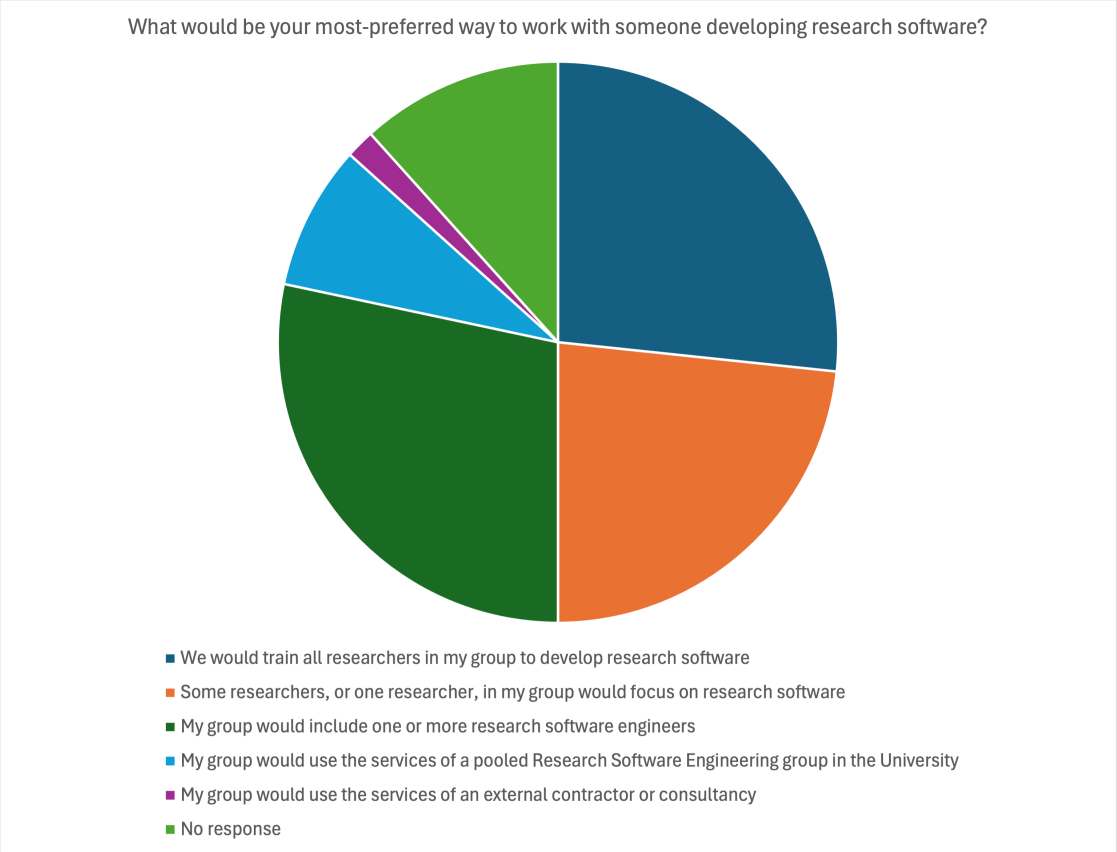


Figure 5.9: Responses to the question "What would be your most-preferred way to work with someone developing research software?"

their relationship to the professor. We therefore consider the high representation of students among our respondents to be a selection bias.

On the other hand, six staff responses—0.03% of the University staff—is a disappointingly low "turnout": even when we note that the 16,000 figure cited above counts *all* University staff, and we recruited among research staff and research software developers, who are a large fraction of that cohort. We hypothesise that the long hours culture and intensification of work in UK academia, as reported by Sang et al. [178], reduce employees' desire to participate in additional voluntary activities including survey-filling.

The responses to our survey are significantly weighted towards the scientific faculties (97% of all responses) and students (90% of all responses). The small number of responses from social sciences and humanities divisions seems to represent an absence from the sampled population, as we sent the survey to both broad

cross-functional distribution lists and to members of a digital humanities support group. We hypothesise that research software engineering—both in the formal sense of people and groups named RSE, and the informal sense of applying software engineering techniques to research software—is more mature and widely adopted in the physical and medical sciences due to its genesis in the e-science field. In section 5.3.2, we propose additional work to test this hypothesis.

The design of the survey makes it impossible to distinguish responses from researchers in departments in the same division; for example, responses from the Department of Archaeology and the Oxford Internet Institute would both report their division as Social Sciences, even if the departments have very different levels of engagement with research software. We acknowledge that this is a weakness of the survey design that we did not consider when we adapted the survey Dr. Robinson made for gauging the "business case" for an RSE group in the university, in which it makes more sense to organise responses by administrative division. In the event, unfortunately, the low turnout means that finer-grained analysis would not be possible even had we asked respondents to indicate their departments.

We see that almost all respondents (58 out of 60, 97%) spend some time on using or developing research software, and that 100% of respondents think that research software is at least somewhat important to their work, with more than half (34/60, 57%) considering it vital. Comparing these results with Hannay et al.'s 2009 finding that scientists spend around 30% of their time developing scientific software and that 84.3% of scientists believed software to be important or very important to their work [105], we find that the importance of research software has increased. The figures on time spent are not comparable as our survey asked about using or developing research software, while Hannay et al. asked only about development.

Despite this importance, self-teaching is still a common pathway to acquiring software engineering skills (30/37 responses, 81%) though it is typically blended with formal education or mentoring; only 5 respondents (14%) reported that their skills were solely self-taught. When mentoring is involved, the mentor is more frequently a researcher (12/37, 32%) than a research software engineer (2/37, 5%).

This leads us to answer RQ1 negatively; almost all researchers need software skills, and most learn at least partially from self-teaching and being mentored by each other, so training is not consistently available.

However, when respondents collaborate with others on their research, they more frequently hire RSEs (13/21, 62%) than any other collaborator. Additionally, the most-preferred way to work with someone developing research software is to include RSEs in the research group (17/54 responses, 31%), though the nearest alternative preferences are almost as popular: training all researchers to develop research software (15/54, 28%); and a subset of researchers focusing on research software (14/54, 26%). This potentially points to software engineering being perceived as an expert skill that should be delegated to specialists (either RSEs or focused researchers), rather than a necessary competency for working in research; this despite the importance that the same respondents place on research software.

In section 9.2 we compare the results of this survey with the qualitative research we undertake in the following chapters.

5.3.1 Threats to Validity

By recruiting members of a single institution (the University of Oxford), to complete the survey, we took advantage of a diversity of backgrounds, disciplines, experience levels, and career pathways at the cost of a non-diverse institutional context. The results of our survey may not reflect the population of UK researchers, if Oxford presents a unique institution in which to engage with research software, or a single point in a hypothetical "space" of software engineering maturity. However, researchers tend to change institution throughout their career without a strong "loyalty" to their current employer [176], so a sample drawn from one institution still includes views of people who have experienced many others.

The low number of responses, and bias towards respondents in the Mathematical, Physical, and Life Sciences division, mean that the survey results may not be representative of the population of researchers in the university.

5.3.2 Further Work

This survey provides quantitative insight into the state of research software engineering in Oxford University. The results suggest various directions for qualitative work, some of which we explore in the following chapters.

Having identified how researchers and RSEs work together, and the kinds of software they work on, interviews with researchers and RSEs can illuminate the details and provide individual examples of the macroscopic trends observed in the survey results. We conduct and analyse a series of such interviews in chapter 6. Additionally, in chapter 8, we administer and analyse a series of focus groups that allow us to observe the interactions between researchers and RSEs in a research group as they negotiate their work.

Collecting data from people with a broader range of institutional backgrounds and experiences would mitigate the effects of sampling survey responses from a single university. We explore the effects of institutional contexts and demands on individual actions in chapter 7, and the interview and focus group participants in chapters 6 and 8 are drawn from a wider UK academic and commercial context.

A particular feature of the survey results is the low representation of respondents in the arts, humanities and social sciences. The Software Sustainability Institute has surveyed the Arts and Humanities Research Council community to understand their views on software and digital humanities [106], and further work could explore the connection (or lack of connection) between the research software engineering provision available in universities and the digital humanities and social sciences.

While we have some indication—through the responses to question 7, reported in figure 5.3—of the kinds of research software our respondents are developing, our list of fixed responses doesn't map onto any of the ontologies or categorisations of research software discussed in section 2.2.1. Future work could explicitly ask people who write research software to locate their software in one of these systems, or ask them to categorise their research software in an open-ended way, and compare the socially-constructed typology that arises with the literature.

In addition to deeper research into the field of research software engineering, research on how researchers themselves perceive research software could elucidate why a majority of researchers would not prefer to train all of their colleagues in software engineering practices.

5.4 Conclusion

While we approached a broad community of research staff and students across the whole of Oxford University to survey their views on research software, almost all responses came from physical and medical scientists, with one response each from the Social Sciences and Humanities divisions, and none from Continuing Education or Gardens, Libraries, and Museums. This suggests a limited overlap between the research software and digital humanities communities.

We found that among people who did respond, research software is universally important to their work, and almost all respondents themselves spend some time developing research software. Despite this, self-learning is still an important route to skills acquisition, and very few respondents have received mentoring from a research software engineer. Additionally, more respondents would prefer to "outsource" research software work to RSEs, dedicated researchers, or other staff than to ensure that all researchers are trained in software engineering.

We propose multiple avenues for further research based on these results, and pursue three of these possibilities in the following three chapters of this thesis.

6

Service or Subservience — Interviews with Research Software Engineers and their Peers

Contents

6.1 Introduction

Having identified the functionalist properties of a profession as concepts for understanding RSE as a profession and the development of the organisations that support

it in section 3.6.1, we interview people who work as RSEs and their collaborators to explore the profession and its properties, its relationship with the existing academic social order, and the values and beliefs of those who work in it.

We start with an exploratory phase of data gathering to get a broad sense of how the people who engage with RSE see the occupation and their positions within it. We thus initially connect our inquiry to the four properties of a profession identified in [171], identifying specific manifestations of the general properties described:

- A body of knowledge for RSEs distinct from adjacent roles such as research assistant, data scientist, or commercial software engineer;
- Technical autonomy for RSEs to define how research software is designed, developed, and maintained;
- An idea of the purpose or value of RSE beyond fulfilling immediate needs on any particular research project; and
- Evidence of some material or social benefit for those who fulfil the role of research software engineer.

Exploring these attributes of RSE allows us to explore the "boundaries" of the occupation: whether people see these attributes as present for RSEs; whether they perceive their existence as distinct from, or bound up with, other occupations; and the activities and processes that reproduce or weaken these properties.

6.2 Method

6.2.1 Designing and Conducting Interviews

To develop an understanding of the RSE occupation and build an answer to RQ2, we conducted a round of open-ended interviews, with prompting questions based on the functionalist properties of professions. Participants were initially recruited via a post to the Society of RSE Slack workspace, and subsequently via theoretical sampling—relying on the data from the interviews, including mention of other

roles or stakeholders and directly asking participants, to uncover gaps and identify potential recruits. The recruitment message and interview script are reproduced in appendix A.

The respondents were all based in the United Kingdom, which is consistent with our decision to constrain the scope of the study to RSE in the UK. In addition to the pragmatic benefit of reducing the scope of the problem we seek to address, we also remove some variances between geographies that would have confounded our analysis. For example, the funding models of academic institutions in the UK is very different from the approach employed in the United States, with a greater dependence on state funding (both tuition fees and research councils).

There were thirteen interview participants, though one subsequently withdrew from the study and their data are not included in the analysis. The participants are:

1. An RSE working in an RSE group at University A.
2. An RSE working in an RSE group at University A.
3. A project manager in a high-performance computing department at University B.
4. The editor-in-chief of a data science journal published by a large academic publisher.
5. The leader of an RSE group at University C.
6. A software engineering group lead at a government research institution.
7. A senior RSE working in an RSE group at University A.
8. An RSE working in an RSE group at University D.
9. This participant withdrew from the study.
10. An epidemiologist working in a medical data science research group at University A.

11. An open source infrastructure engineer, working for a company that makes tools used by RSEs.

12. A high-performance computing specialist at a publicly-traded multinational software company.

13. The leader of a research computing group at University E.

Universities A–E are all in the Russell Group. Participants were interviewed by the author on Microsoft Teams, with interviews recorded and transcribed with participants' permission. The author then reviewed and corrected the transcripts against the recording, then anonymised them by removing names and other identifying details. The anonymised transcripts were then loaded into NVivo for coding. Following the constructivist paradigm, the transcripts were coded sentence-by-sentence, making note of the ideas, events, and feelings mentioned. Subsequently *axial coding* was performed by comparing the codes across the different transcripts, grouping related codes and identifying abstractions that encapsulated the groups. The codes and abstractions were then associated with the properties of professions described in the functionalist model to discover if, and how, the participants in RSE engage with the definition and advancement of RSE as a profession.

Participants were additionally invited to optionally attend "participants' council" meetings, facilitated by the author. We presented the current state of the research at the (online) meetings of this council for sense-checking, and to allow participants to raise any questions or concerns about the research and its application. These discussions were conducted under the Chatham House rule [206]. We noted that discussions were lively and engaged multiple participants, often involving a text "back channel" in the meeting chat alongside the audio-visual conversation. Our notes from the meeting and the anonymised chat transcripts were also collected and coded as part of this study.

6.2.2 Constructivist Approach

Our study adopts a constructivist paradigm, as described in section 4.4. We build a picture of the socially-constructed reality that is RSE by engaging with the people who construct it and the events that shape their opinions of the field, their work, and the intersection of their identities with the concept of Research Software Engineering. We avoid defining any of the properties of professions before asking participants about them. This enables the participants to bring their own definitions and apply the terms to their work as they understand them. Where clarity is needed, we invite participants to expand on their answers to bring more detail to their interpretation of the questions we posed.

As an example, consider this exchange from our interview with Participant 1:

> [Graham Lee] Uh, so does research software engineering, have a clear body of knowledge that engineers are expected to understand?
>
> [Participant 1] No, I wouldn't say so. It's it's a very inclusive field, which is nice, but that also means it can not be clearly defined in some places, especially as a new and evolving discipline.
>
> [Graham Lee] I did, I mean, do you think they're very are say, unofficial or ad hoc, um, ideas of what an RSE is expected to know, as distinct from someone else?
>
> [Participant 1] Yeah yeah, yeah. Sure, and that that can vary depending on who you ask. Yes, uh, so the definitely there is there is there is an idea shared understanding of what an RSE is, but I don't think it's been like codified or written down as I know of.
>
> [Graham Lee] Do you think you'd be able to give like your view on what that shared understanding is?
>
> [Participant 1] Sure, I think I mean in kind of work I've been doing and my colleagues. I think research software engineer is...

The participant interprets the phrase "body of knowledge" as a reference to an unvarying definition of the field, when the question is first posed. We offer an expanded meaning, in which the body of knowledge is shared but not codified, and the participant offers a detailed answer within this expanded context. The exchange therefore moved from a very closed response (there is no body of knowledge) to a rich description of of the body of knowledge, embedded within a interpretation in which the participant sees codification as an opposition to the properties of inclusivity and evolution.

6.3 Results

The position of RSEs in their professional context can be understood from two perspectives: what the RSEs themselves think of the role, the people who occupy it, and the interactions with colleagues who adopt different roles; and the perceptions of those who work with them. In the latter case, note that the RSEs who participated are also reporting on the perceptions they have of how their colleagues see them, which gives more insight into the self-identity of the RSEs making the statements.

The RSE self-perspective can be further broken down into the work that RSEs report they do, and into ways in which the role or identity of Research Software Engineer is defined and bounded. Examples of role-defining statements would include assertions participants make about their own identity:

> [Participant 10] So sometimes I do say I'm a software developer. I very rarely say it's research software engineer.

and those they make about whether other people or groups of people should be considered research software engineers:

> [Participant 7] I also discovered that those people who have stable jobs and do sort of RSE kind of things and they are in stable positions. They don't call themselves RSEs, they don't know they are RSEs, and they're not even interested in that at all. 'cause they have [...] themselves sorted out.

In the following sections, we present data gathered from the interviews through the lens of the four properties of a profession, listed in RQ2. To answer this question we look for evidence of the RSEs and their colleagues interacting (or not) with the four properties as they discuss their work.

6.3.1 Body of Knowledge

There wasn't a shared model of the knowledge required by RSEs in their work. Participant 6 suggested that the knowledge requirements were "tacit" in the hiring decisions made within their group, and suggested we look at the "selection criteria" they used in the recruitment process to understand the knowledge expected. As shown in section 6.2, participant 1 agrees that the body of knowledge is part of a "shared understanding" of RSE that isn't "codified or written down". They say that an RSE is someone "conversant with" software engineering principles and able to apply it in their work; they also "ideally" have enough domain knowledge "to converse with academics", but accept "that's not always the case".

Knowledge of software engineering is also evidently expected of RSEs by most of the other participants we interviewed. In addition to specific tools and technologies, multiple participants say generally that RSEs should understand the software engineering body of knowledge[1], for example:

> [Participant 13] Basically, the software engineering body of knowledge as uh developed within, you know, within sort of core engineering, not for or in numerical disciplines and. but I think you know just getting the core software engineering body of knowledge across. Uh, all the research disciplines where you know many significant codes are still 10,000 line single function FORTRAN codes is enough to be going on with.

This participant was the only one to extend this idea to specific numerical programming concepts relevant to the sciences, with "methods of uncertainty

[1]We note here that no participant who says RSEs should understand all of software engineering enumerates the knowledge areas that comprise "all" of software engineering. In chapter 7 we turn to published sources to compare the idea of software engineering as practised by RSEs with software engineering as described by that field's professional bodies.

quantification and sensitivity analysis". However participant 6 also noted the recent creation of an "intermediate scientific software with Python" training course, again demonstrating that there are other sources of information we can use to infer the RSE body of knowledge.

Opinions on whether an RSE needs to understand the research domain they're working in are more varied. In the extract from participant 1 above, we see that domain knowledge is "ideal", but "not always the case, and that's fine". While Participant 8 says it gives them a "user perspective" on whether the software they have been asked to build is appropriate, participant 2 says that this "helps" but that they are "not required to have a deep understanding of any particular research question".

Participant 12 accorded with Participant 1 that both research and technical components are "ideal", simultaneously identifying the two separate contributions while also situating both in the RSE role.

Further evidence of the vague definition of the body of knowledge is the lack of either a formal mechanism for training an RSE, or a common pathway into the role. Multiple participants report that learning on the job—perhaps through a formalised mentorship but not necessarily—is an important part of the role. To participant 12, "the best RSE is going to be one that is maybe doesn't know everything, but at least is able to learn and is willing to learn and is interested in finding things out and reading". Participant 1 confirms that when it comes to learning RSE techniques, "it's an ad hoc method, you know, through osmosis", and Participants 3 and 5 agree that they picked up a lot of skills "along the way", with Participant 5 saying that this broad (and vaguely defined) body of knowledge could—and perhaps should—be formalised into an education programme that would serve as an entry to the RSE career.

Participant 1 agrees that a course for RSE "ideally will cover all the basics, so definitely various sections like the software engineering bits", including "ethics

issues". This participant was one of three—two RSEs and an infrastructure engineer—who mentioned ethics during their interviews, but the only one who said that ethics was something that RSEs needed to learn.

6.3.2 Autonomy

Participants report very different experiences when it comes to the amount of autonomy they have in their work. One, a RSE in a university RSE group, reports that the freedom to choose how the work gets done is ultimately granted by the principal investigator (PI), and that they enjoy the occasions when the PI enables that freedom.

> [Participant 8] I have had experiences that have really allowed a lot of autonomy and creativity, and that's why I really like that's why I enjoy it and so the types of project I've worked on.
>
> ... there's lots of times where. The PI says things like have a think go away and. Have a think how how you want to do that or propose some ideas or what do you want to get from it or you know you know is there something that for your career? You want to try out? I don't think that's nec.... That's not necessarily all projects, but also you know going back to what I was saying before about, I think you'd get really different answers from my colleagues.

Participant 1—also employed as an RSE—disagrees, saying that the PI lets "you do really what you want as long as long as the project works". Conflict over technical autonomy comes from working with other RSEs: "in any of the large projects and some of the technologies you know you may disagree on what technologies to use". For that participant, changing technologies is both a challenge and a source of excitement.

Two other participants connect their relationship with other RSEs with the idea of technical autonomy. Both link seniority within their RSE team to the amount of freedom they have. Participant 10 reports that, as the person considered to have the least technical experience on their team, "I would say that I probably get stuff checked more because they obviously don't want me to do something completely crazy. Uhm, and like reimplement something in Rust or do something like mad".

Participant 3 is the only person who explicitly associates seniority with autonomy through their organisation's salary structure: "for a standard grade 7 for an application developer you would usually be expected to attend meetings with external people, but there's usually be somebody more senior there from [Participant 3 Department] to support you and make sure you're not left hung out to dry whereas, and a Grade 8. You'd be more likely to be expected to go in and be able to hold your own and provide technical advice without supervision". This participant is also the only one who explicitly links seniority with their grading structure (and therefore with autonomy), and goes on to explain that the difference between grade 7 and grade 8 is "not really technical differences right? They're the more on the sorts of responsibility mentoring [...] management side of things."

Whether the senior RSE actually has technical autonomy is said by one manager to depend on whether the PI is interested in software or not, providing a more nuanced view on the relationship between an RSE and a PI expressed earlier by participant 1; if the researcher has a software interest, they may be more inclined to get involved with the technical decisions.

Autonomy and Academia

The examples above show that autonomy is sometimes perceived as a "gift" that can be bestowed, or not, by the PI on a research project, associating intellectual freedom with a position in the academic system. Participant 3 self-identifies as a "senior RSE", though they have a different job title ("project manager") to align with institutional expectations, and describes advantages to being outside the usual academic group structure. When RSEs are employed by a central software group, they can focus on delivering software without having to deal with bad management practices in academic groups, who can treat people "really, really horribly".

They are the only participant who describes this centralisation as a benefit. In opposition, participant 13 reports that centralisation can be culturally negative, likening RSEs in a centralised IT department with a character from the sitcom

The I.T. Crowd who only gets to ask whether an academic has "tried turning it off and on again".

We note that the literature supports a relationship between professional autonomy and supportiveness of the workplace, with Goodboy et al. reporting strong negative correlation between decision authority and workplace bullying among US academics [207], and Franco-Santos et al. finding positive correlation between collegial governance practices and employee wellbeing in UK academia [175].

A further example of the limitations of autonomy within the academic sphere is given by participant 7, who identified that "since I started working as an RSE that just that, that possibility [to spend time on training and professional development] just vanished." They say that RSEs "are seen as tools to some extent", but that "we are still thinking tools and embedded in a research question we would come up with things that could make that research better".

Participant 4, editor-in-chief of a data science journal, indicates that the limitations on autonomy can come from cultural norms. Requiring that authors publish their software to a repository before a paper is accepted could be perceived as putting "barriers up to our authors to publish with us". The requirement "potentially [does] not fit the [...] domain specific requirements that are out there", particularly when "there is code that is dangerous because it could be used for hacking important systems".

The theme of domain-specific publication restrictions is not found in other sources. For example, the University of Bristol's Advanced Computing Research Centre recommends authors publish their software "only if you really want it to be used by others, and you have the time or a plan to support it post-publication" [208], but they do not discuss dangerous code, or domain-specific expectations. MIT Libraries do not describe any caveats associated with publishing research software [209]. The FORCE11 Software Citation Implementation Working Group's Journal Task Force journal production guidance for software and data citations notes that authors should indicate where a registration and/or fee is required to access the software, but no domain-specific expectations or reasons not to publish research software [210].

These recommendations to avoid publication, whether because the software potentially has a dual-use nature, or to avoid support overhead, are inconsistent with the recommendations to use and publish software as open source seen in section 3.2.3.

An Autonomous RSE Community

While RSE has special-interest groups and community societies that offer conversation, support, and education, these organisations don't fill the role of a professional association that sets standards, confers professional status, or otherwise "polices" the professional boundary; a sign of an autonomous profession. Multiple participants discuss the history and position of the social nature of RSE, or their positions within it. From these discussions, we see that the growth of an RSE community has provided practitioners with a sense of belonging and support—and that this growth has been explicitly driven by a small number of actors.

For example, participant 3 says their department—a High-Performance Computing centre at a Russell Group university—was traditionally seen as "a bit aloof and separated from the rest of the UK HPC community", until that participant became a trustee of the Society of Research Software Engineers and used that position to "integrate" their department into the community.

Because we recruited participants from the Society of RSE Slack chat service, responses will be biased towards people who are more socially active in that community. People who are RSEs but don't engage in the community at all—for example, proto-RSEs—are not represented in our data.

6.3.3 Service to Others

Some of the RSEs interviewed seek to position themselves in a liminal space between "technician" or "IT support" roles that are associated with the depth of their technical contributions, and "researcher" roles that provide a peer status with academics and recognise their intellectual contribution to the research activities.

[Participant 8] Yes, providing a service but we are maybe more. Along, we're a collaborator as well. There's something there about how I feel like I get to play a role in research projects, which isn't just somebody in IT who, you know, gets given a ticket.

Participant 13 agrees that RSEs occupy a space overlapping IT and research, and that this duality is key to the identity. "A really important part to me of the research software engineer identity is that we are proud to be IT professionals and proud to be scientists and we are both of them at once. That's the point."

Participant 5 wants RSE to be a service to researchers but identifies a problem of scale:

[Participant 5] Um at first and we thought we would be going out and like, uh, you know, helping software developers write software and making our software developers have time available to researchers and I quickly was just not scalable enough. You know, 210,000 researchers and 8 of us. Won't work.

Participant 11 puts this service to researchers on a tangible footing: making it easier for researchers to build on existing work that has a computational component. In doing so, they point to the importance of open source software in avoiding ensiloing research software, a point also made by Fortunato and Galassi [122].

6.3.4 Privileged Status

Status within the academic community

A common thread is the idea that RSE occupies or creates a fuzzy boundary between the existing roles of researcher and software engineer. But is an RSE a researcher with a specialism in software, or a software engineer with special knowledge of research projects? The answer would seem to be "both", although when discussing project management there seems to be a clear tendency to refer to an external "researcher" as client, for whom the RSE is acting as an agent. For example, Participant 13 describes the frustration when "the researchers" changes the requirements, saying they "don't know what they want" and create vague or ambiguous requirements

that are difficult to satisfy correctly. Participant 2 is more phlegmatic about this situation, describing similar cases in which researchers realise that the software isn't what they expected but identify the cause as their own insufficient clarity.

Participant 6 similarly externalises the "research" aspect of the work, saying that a good model for a project has "a sort of lead RSE in part equal partnership with the sort of lead researcher." However, they find cases where "that lead researcher is actually. Quite software focused themselves and so that division isn't quite so clear." This affects both their status relative to the researcher, and the amount of technical autonomy they can expect.

Participant 7—an experienced senior RSE with a background working on software in multiple research domains—draws a distinction between being engaged as a software engineer on a project and being a researcher, even though the activities may well be the same. This potentially indicates a conflict between the goals of research software engineering and the implementation: the aim to be seen as peer contributors to research might be undermined by highlighting the distinct technical contribution.

Having learned from Participant 7 that a researcher can have a lot of responsibility for creating software without explicitly being engaged as an RSE, we look for evidence that RSEs see their role as distinct from that of a research assistant or researcher. This same participant reinforced the position that there is no difference: "My view is that RSEs are also academic."

For a lot of the participants, this question was closely tied to ideas about the culture of academia, particularly a "lone genius" or "lone wolf" model of academia that sees the principal investigator as a solitary genius inventing new science, and the people around that genius as subservient. One RSE department head identifies cultural resistance to attempts to increase the team's diversity by hiring applicants without a traditional academic background:

> [Participant 13] We still have a credentialing issue with our academic collaborators, who tend to look down on you and sneer a bit if you haven't. Uh and I, uh, gone through that pathway

> and then we see, you know, I mean we're we're. So we have sales work to do to the academics, which pushes us back down this sort of traditional, snobbish kind of academic cultures.

They explain "what's motivating us to realize that a lot of the reason why scientific software has tended to be not. Created with the quality and care that we think it deserves is because of the incentive structures within academia that prevent you from making the time to to to do that well",

Participants pointed to examples of similar technical roles that they perceive as successfully carving out niches as respected parts of the academic community: software scientists and data scientists. Participant 7 suggests that "RSE" is an ad hoc social change organisation or unofficial trade union of people in software-related roles who don't have stable positions, and that the other, non-RSE titles are adopted by people doing the same work with better conditions.

Participant 13 groups RSEs and data scientists as "digital research professionals" with other "research engineering roles like the engineers that build experimental facilities" into a middle field of "research professionals" who straddle the categories of academic and professional services staff, reinforcing the view of RSE as a liminal role.

Status relative to software engineering

The fuzzy boundary between research and RSE can be compared with the fuzzy boundary between software engineering and RSE. Participant 10 says "obviously the coding is similar because it's Python coding in general that we're using in both", and tends to use the phrase "software developer" rather than RSE to describe their work: "I work in a research field, but I could very like I could be writing Python for a startup."

Participant 3's department works with both commercial R&D and academic research and says "actually the work we do for commercial people does not differ very much [...] for from what we do for academic people." Participant 2 agrees that RSE and industry software engineering are "a very similar job in a different environment."

Differences between research and commercial software engineering are also brought up. The project manager who reported that their commercial and academic work is very similar also says that commercial organisations are much better at:

> [Participant 3] Setting up the agreement on and just being very clear about who's role's what, who's going to deliver what when right? Commercial group entities are much better at that sort of interaction, I think.

Participant 2, who described the two jobs as very similar, says that there is much less prestige in industry. Where this participant sees the academic perception as "these are important people we need to help us accomplish our our interesting research project," in the commercial sector "it's more like all those are the strange people who make stuff happen."

Two participants say that industry software engineering has better process management, with participant 3 particularly pointing out there's usually much more defined processes for dealing with [changing requirements].

Finally, two participants bring up the distinction in salary between academia and industry, with participant 2 describing commercial software engineering salaries as "2, 3, 4 times what you get paid in academia" and participant 10 agreeing that RSEs are "earning a third of what they could earn in industry."

6.4 Discussion

We observed that RSE has a liminal existence, in which responsibility, authority, and autonomy for tasks and outcomes in the computational research domain are not clearly delineated. The majority of these conflicts are at the intersection of RSE with academia, where a "snobbish" culture will "look down on you and sneer a bit if you haven't gone through" the traditional, hierarchical academic pathway (Participant 13). This liminal occupation features no clear definition of the role or responsibility from members of the profession, its community, or collaborators, beyond the almost tautological observation that they are somebody involved in both research and in creating software.

Even participant 6, whose role in a government-funded research facility doesn't put them in direct contact with academic culture, makes the observation that this separation makes it easier to deal with "the kind of academic egos thing". Meanwhile, participants do not describe problematic relationships with other stakeholders, including with funders who are described as generally supportive and even willing to "take a punt" on initiatives that support sustainable software.

Noting that the funding councils draw their pool of reviewers for funding proposals from the academic population, and therefore that the same people are described as both supportive and antagonistic, we suggest three possibilities. Firstly, that the structure of academia leads to the observed behaviour; in other words, that the people themselves are not (all) "snobbish", but that the system in which they operate leads to actions that are perceived as derogatory towards people who support their work. Secondly, that the academics who are more supportive of others' work adopt positions such as funding reviewer that enable such work. Thirdly, that some of our participants are frustrated at the difficulties of navigating the academic career path, and infer an antagonistic relationship to justify the perceived setbacks.

Some people say that publishing academic papers is important, others that it is off-putting and should not be part of the RSE's responsibilities. Sometimes, the RSE is a researcher who happens to have a computational bent. At other times, they are an agent engaged by a researcher client. In many ways, the role is seen as similar to commercial software engineering—but important differences are also reported, particularly in relation to perception and pay.

Despite the many ambiguities offered, there does seem to be a strong sense of connection to RSE as an identity, both individually and collectively. Practitioners indicate that they have adopted the title of "Research Software Engineer" unofficially, or would like to be able to use it. The identity is correlated with a struggle for recognition: leaders report on the importance of the identity in gaining recognition for the contribution made to research by people writing software, and one practitioner said that people who have got stable roles doing the same work don't use the label RSE.

The inputs that can change the effects of these conflicts and ambiguities remain hidden: for example while participants reported that the "snobbish" academic culture limits the status of RSEs, we have not seen what reinforces or changes that culture and how the campaigns to raise the status of research technical professionals impact the community. Investigating these further would help to reconcile—or expose differences between—the views of the RSE leaders and the practitioners: is RSE solving the problems that the SSI collaborations workshop participants set out to solve when they came up with the term? It would also provide a practical model to guide the development of the field, allowing RSE leaders and practitioners to understand what they can change and how those changes are connected to the system of academic software construction.

Some participants did recognise that the RSE field is not unique in its blending of responsibilities across the traditional dichotomy of academic or non-academic identity, comparing their position with that of data scientist or software scientist. There is a broader range of "blended" roles with which comparisons to RSE could be made, which would provide case studies and guidance to RSE leadership in defining a "third space" career pathway.

All of the participants who are RSEs or work with research software discussed work in scientific disciplines. Our participants included practitioners from multiple institutions, and we avoided targeting specific disciplines during recruitment. This is consistent with our finding in section 5.2 that most RSEs are active in the sciences. We connect the findings in this chapter with the other research in this thesis in chapter 9.

Comparing our participants' roles with the stakeholders we identified in section 2.2.3, we have engaged with RSEs (some of whom have been proto-RSEs and researchers during their careers), a journal editor, a researcher, software producers, and RSEs or group leaders who also have positions in support organisations. We did not interview any current proto-RSEs; students; researchers in software engineering; people who represent RPO management, or administration; people who represent funding bodies; or users of computational or software-supported research.

As such, our interview data focuses on the practice of research software engineering as seen by the practitioners, and doesn't explore how the (potential) profession of RSE is perceived by people who hire RSEs, provide workplace training and career development; or people who might enter the RSE career—or, crucially, might choose not to.

6.4.1 Towards an answer to RQ2

We asked the question "Does RSE exhibit the four properties of a profession: a body of expert knowledge; technical autonomy; valuing service to others; and privileged status?" These properties are exhibited in part, not always clearly, and not necessarily in distinction from other occupations.

There is no agreed-upon body of knowledge for RSE. People make reference to external bodies of knowledge, for example the software engineering body of knowledge and the expectations of researchers in their research domain, but without consistency.

We saw that the ideas of autonomy, service, and status all seem to be caught in the struggle of "service or subservience", particularly with relation to a more traditional academic research occupation. The "fuzzy boundary" between research and RSE means that some see RSEs as researchers with an interest in computation, while others see them as agents working for researcher clients. Removing RSE from the typical academic structure into a separate group can help avoid poor management observed in research groups, and help RSEs to focus on delivering software, reinforcing the agent role. But that organisation is not universally implemented or desired, with some RSEs describing the relationship with researchers as a peer role within the same group. Others see the autonomy and status of the RSE as entirely at the whim of a principal investigator.

6.4.2 Participants' Council

The participants' council meetings acted as a sense-check of our analysis of the interview data, as we presented the results from our analysis to the participants

and invited their comments, feedback, and criticisms.

One participant in a council meeting noted via chat message that RSEs should compare their situation with "data scientist" or "software scientist" roles, which have fewer hierarchical implications but "often similar/overlapping roles with RSE".

Attendees at one participant council meeting agreed with the theory we presented from the interview data that academic norms create a two-tier structure in which contributions other than primary authorship of scholarly papers are seen as secondary or even subservient. One attendee (not identified here due to the Chatham House Rule in place at the meeting) said "Upstairs/downstairs is definitely the phrase", referring to the class division in British stately homes between the wealthy landlords who live upstairs, and the working-class servants living and toiling downstairs. Another participant in the council discussion questioned whether there should be a separate term or title for RSE in academic parlance at all, as it "reinforces the hierarchy and causes conflict". This notion of RSE as a challenge to the clear distinction—and associated power imbalance—between academic and non-academic identities fits into a historical context in which lines have been blurring between the responsibilities and career trajectories of academic and non-academic staff since the 1990s [211], leading to a rise in "blended" or "third space" roles that are neither clearly academic nor non-academic [176]. Nonetheless, it can still be difficult for technicians to transition into academic careers, being "typecast" as technicians within their employing institution [212].

In one of the participant council discussions, the "lone wolf" view of academia was described as fragmenting the RSE community, as the drive to identify as a significant contributor in a particular research domain may dissuade people from taking other professional identities. This evidence weighs against the idea that the RSE identity conveys status in the academic context, as participants must adopt other identities to achieve recognition in their fields.

6.4.3 Threats to Validity

In this work, we spoke to a small number of participants: data from 12 people are included in the analysis, of whom 7 perform RSE or related roles, 3 are leaders in RSE groups or organisations, and the remaining two are in roles where they interact with RSEs. This is a small number of participants whose experiences and perspectives may not be reflective of the field in general. In particular, our results primarily reflect an "insider view" of the field, as only two of our participants engage with research software engineering as non-practitioners; we see something of other stakeholders' interactions, but not from their own perspectives. The outcome of this work is refinements to the research questions and future directions to explore, meaning any conclusions drawn at this stage are necessarily contingent and subject to further testing.

The author spent three years working as a senior RSE, including working with some of the participants, as part of a 20-year software engineering career. This experience has been invaluable in interpreting the views of the participants including the technical argot in use both in software engineering and in academia, but could also be a source of bias in the investigation, particularly when considering the social constructivist axiom that the "truth" in the research is a result of the interactions between all people involved, including the researchers. To address this source of bias we have used a reflective methodology, frequently writing memos on the research as it progressed and exploring how our own experiences and expectations had entered into the analysis.

6.5 Conclusion

While the Society for Research Software Engineering has a clear mission, which aligns with the purposes identified for creating the Research Software Engineer identity at the 2012 SSI Collaborations Workshop, the current behaviour and expectations in the field of RSE are ambiguous and plural. The role appears to occupy a liminal space in the shadows of academia, in which practitioners see a distinction between their

goals and those of "traditional" research assistants but do not necessarily see that this distinction leads to differing expectations from academics or their institutions.

The body of knowledge expected of RSEs varies, depending on who is asked, from a broad ability to figure things out to a detailed knowledge of the software engineering body of knowledge and numeric programming and verification techniques. Of the four properties of a profession we sought to investigate in relationship to RSE, none is unambiguously present. Practitioners talk in general terms about the degree of autonomy they have, but provide specific examples of collaborators on projects curtailing that autonomy. The RSEs agree that their work in creating software is in service to the research, but are careful not to portray themselves as subservient to researchers: indicating that ideas of service and of status are intertwined and often precarious.

These early findings offer lots of avenues for future work. Given the liminality observed in RSE, the fluid relationship between RSE, academia, and software engineering, and the sometimes conflicting requirements they impose on practitioners, we turn to neo-institutionalism to address the remaining research questions listed in section 4.2.

Neo-institutionalism portrays professional groups as institutions that struggle among other institutions for survival, based on their relative legitimacy [189]. Such legitimacy can be pragmatic (i.e. it's in an individual's self-interest to treat the institution as legitimate), moral (the institution's logic accords with normative values), or cognitive (it's easier to treat the institution as legitimate than otherwise) [187]. Institutions embody "logics" that lead to particular actions, and the legitimacy of an institution lends weight to the choice to select its logic [183].

In the following chapter, we explore the institutions and logics relevant to RSE to seek an understanding of the occupation's relationship with neighbouring disciplines and the extent to which it has acquired or is developing an institutional legitimacy within its domain of relevance. This approach naturally follows from the results of this chapter, that indicate ambiguous legitimacy overshadowed by the needs and demands of researchers.

7

Performing the Role in Public — Blogging About RSE

Contents

7.1 Introduction

Having established in Chapter 6 that RSE adopts a liminal position in the shadow of academia, and that it shares a lot of expert knowledge with the domain of software engineering, we wanted to explore the relationships and boundaries between RSE

and these other fields. Understanding how RSEs navigate the relationship between academia and RSE, how RSEs use, ignore, or augment knowledge, and what resources and support they use to do so, can help to answer RQ3 about the influencing factors on RSE's work, and the legitimacies competing to influence the field.

7.1.1 Collecting Data from Blogs

We gathered data for this phase of the research from blogs published about research software engineering on the Internet. This provides a very different context for the collected data than the interviews conducted in Chapter 6. Blogs are deliberate creations by their authors for public propagation, so the blog authors present their ideas in ways that they consider suitable for broad dissemination. Contrast this with the one-to-one conversational nature of interviews, in which participants may be more willing to share aspects of their work that they wouldn't consider publicly sharing under their name. Additionally, where interviews are co-created by the interviewer and the participant, the text in a blog post was chosen by the blog author without any interaction from the researchers.[1] Kurtz et al. report that blogs represent an avenue where people perform diverse physical, social, and emotional identities that is relatively unexplored by ethnographers, but that "an ethnographic approach to web-based research is generally successful in documenting and analyzing data extracted from blogs" [213].

7.1.2 Neo-Institutional Theory

Introduction to Neo-Institutional Theory

To investigate how RSE is performed and presented in blogs, we adopted a neo-institutional approach, outlined in section 3.6.5. We selected NIT for this study as the model of competing institutions and logics it proposes accords with our research question on "factors" affecting RSEs' work, and how they negotiate those factors.

[1] The named blog author may have incorporated other contributions, for example editorial reviews, from peers and colleagues prior to publication. Nonetheless, the blog post represents something that the blog author decided to publish about the topic in question under their name.

Given the participants' great dependence on universities and the state research council model observed in Chapter 6, we consider the neo-institutional idea of isomorphic change especially relevant to this work. DiMaggio & Powell hypothesise that certain conditions lead to higher degrees of isomorphism [188]. Particularly, their hypotheses A-1 ("The greater the dependence of an organisation on another organisation, the more similar it will become to that organisation in structure, climate, and behavioural focus"), B-1 ("The greater the extent to which an organisational field is dependent upon a single (or several similar) source of support for vital resources, the higher the level of isomorphism"), and B-2 ("The greater the extent to which the organisations in a field transact with agencies of the state, the greater the extent of isomorphism in the field as a whole"), would appear to be relevant to RSE.

Investigating these processes of isomorphism will help to address RQ3, as they uncover the extent to which RSE is dependent on the structures and rational myths of academia. We would also expect to see a "tipping point" in RSE adoption if mimetic practices lead to isomorphism: as RSE becomes more prevalent in research institutions, other institutions will signal that they, too, perform RSE, because they see it as expected. As DiMaggio and Powell put it, "As an innovation spreads, a threshold is reached beyond which adoption provides legitimacy rather than improves performance." Such performative adoption would not necessary require that the institutions engage with the field in meaningful ways, nor that doing so changes their efficiency or effectiveness: merely that RSE has become part of the field's mythology.

The blog posts examined in this chapter can themselves be seen as institutional work as defined in [214], performed in support of particular institutions and their organisations and ceremonies.

Powell and Colyvas describe blogging as one of the "sensemaking" activities underpinning institutions and their development [215]. "Close examination of organizational archives and correspondence, as well as newer electronic forms such as websites, blogs and e-mail, afford the opportunity to witness organizational performance, and see social reproduction at the micro level, as daily accounts

culminate into ongoing conversations and larger stories about organizational purposes and goals."

They caution not to see individual actors as "cultural dopes" reacting reflexively to macro-level environmental stimuli, and also not as "institutional superheroes" defining reality from a blank slate. Instead, they both respond to and create their institutional realities, sometimes in deliberate generative activities, sometimes attempting to "make do" and understand their work and its context.

In publicly sharing experience reports and promoting particular tools or ways of working, blog authors create and restate rational myths surrounding institutions that they seek to legitimise. As Suddaby indicates, the rhetorical language used in advocating for institutional logics is an important factor in how (or whether) they are accepted and legitimised [184]. Investigating blog posts about RSE gives us a view into how RSE is understood, reported, supported, and shaped by the people who work at it.

7.2 Method

We searched for publicly-posted information about research software engineering, to understand how RSEs present their work and professional identities in public. We restricted our search in two ways:

1. We searched for only blog posts, to ensure that the material we found was consistent in mode of presentation and intended audience; and

2. We considered posts published between January 2023–April 2023, to provide a contemporary "snapshot" of RSE and to pragmatically bound the amount of data to capture and code.

For the purposes of this study, we used a socially constructed definition of "blog": we searched for websites using terms that include the word "blog" and considered any site returned in those results. The results of this search depend partly on self-identification of the websites as blogs, and partly on the algorithmic relationship

between the websites' content and the search engine's ranking algorithm. We only included websites in our data set that have RSS feeds[2], to facilitate automated discovery of articles published within our time window.

To find blogs that discuss research software engineering, we searched for the terms "RSE blog" and "Research Software Engineering blog" using the DuckDuckGo search engine[3]. These search results were retrieved on 20thJune 2023. After initial manual inspection to filter results that were about topics other than research software engineering (including the Royal Society of Edinburgh, and Relationships & Sex Education), we had a list of blogs by RSE groups, organisations, and individuals. We added results from the RSE society link database[4], also retrieved on 20thJune 2023, to create the list of 8 blog feeds listed in Appendix B.

We do not consider whether the blogs are written by authors from the UK or for a UK audience when selecting posts to include in our data set, because the internet is global in scope so RSEs in the UK can read—and be influenced by—blogs from a global community.

We then wrote a script, reproduced in Appendix C, to discover post titles and links from RSS feeds published by each of the blogs between 1 January 2023 and 30 April 2023. This yielded the 17 articles listed in Appendix B, which we saved as PDF files and imported into NVivo. We coded the content of the articles, along with any metadata, such as tags or post categories, sentence by sentence using an open coding technique. We then categorised the codes generated along with the codes in the interview data discussed in chapter 6, to compare the ideas discussed in a one-to-one interview context with those that arise when RSEs share their work and thoughts publicly, in a one-to-many medium oriented towards their peers.

Then we coded the blog data again, using NIT as our basis. Initially we looked for examples of nouns, noun phrases, and proper nouns that represented institutions.

[2]RSS stands for RDF Site Summary, and is an XML format that describes chronological sequences of hyperlinks and metadata in a structured way. RSS feeds are a common feature of blogs.

[3]DuckDuckGo, available at https://duckduckgo.com, is a public internet search engine similar to Google or Bing. The reason for choosing this search engine in particular is the author's preference for their privacy policy.

[4]https://society-rse.org/resources-database/

These could be roles (e.g., Research Software Engineer, professor, project manager), organisations (e.g., university, country, Society of Research Software Engineering), events (e.g., RSE Conference, Collaborations Workshop), or other abstractions (e.g., research career, software engineering, HPC, project plan). We took an inclusive approach, so that almost any noun, noun phrase, or proper noun is coded as an institution. We made this choice to address the challenge identified by Suddaby that it is individual actors, not institutions themselves, that perform the work that establishes and reinforces institutions and supplies legitimacy [184], and the suggestion he and his co-authors made to consider micro-activities in NIT, as explored by strategy-as-practise researchers [186].

For example, in this extract, from a blog post about guides for maintainers of open source projects:

> The Recruiting Financial Sponsors guide stresses the need to diversify funding sources for open source projects to ensure their sustainability. This guide suggests various approaches for identifying potential corporate sponsors, including taking an inventory of current sponsors and project needs, crafting a compelling sponsorship proposal, reaching out proactively to potential sponsors, seeking organizations with aligned missions, and leveraging social media platforms. (Blog 16)

We coded the following institutions: Recruiting Financial Sponsors guide; funding source; open source project; sustainability; corporate sponsors; potential sponsors; social media platforms.

Exceptions include nouns where the author has referred to an earlier idea, but changed the wording to improve the reading experience. For example, in this extract, from a blog post about open source educational materials:

> This mode is already shifting the paradigm in a powerful way—because we can fork this idea. When learners see this style of scientific coding and teaching, they repeat it as they work and share, and they become teachers too. That's what I did, first when I built open data

> science educational resources to teach marine scientists, and then dialed it up when I founded Openscapes. (Blog 8)

We coded the following institutions: teaching; teaching with open source materials; learners; scientific coding; work; sharing; the author's name; open data science educational resources; data science; marine scientists; openscapes. We did not create codes for "this mode", "this idea" or "this style", as these are references back to the idea of teaching with open source materials that was introduced in an earlier paragraph.

We then reviewed the coded institutions and the contexts in which they occurred to discover institutional logics (behaviour that is recommended or deprecated by particular institutions), and evidence of the institutions' legitimacies. We annotated the blog articles in NVivo with our observations on the observed institutions and logics. We also searched for examples of conflicting legitimacies and how the blog authors resolved them.

To continue the example on open source educational materials coded above, we created this annotation to capture the observed logics and legitimacies in the quoted paragraph.

This mode is already shifting the paradigm in a powerful way—because we can fork this idea. When learners see this style of scientific coding and teaching, they repeat it as they work and share, and they become teachers too. That's what I did, first when I built open data science educational resources to teach marine scientists, and then dialed it up when I founded Openscapes. (Blog 8)	Teaching with open source learning materials is powerful because it has viral cognitive legitimacy—when it's the mode of teaching that learners saw, then it's the mode that they will reproduce when they become teachers. The author did this themselves, so they provide their "case study" (going from teaching marine scientists to founding Openscapes, a company based on open source learning materials) as additional evidence of the cognitive legitimacy of open source learning materials (and also the pragmatic legitimacy: it had benefit for the author, so it can also have benefit for you). As their company uses this model, for the author the method also has pragmatic legitimacy: increasing its use increases the market for their company.

7.3 Results

The 17 articles yielded 148 institutional identities, but many of these appeared only on one occasion, in a short mention, that didn't demonstrate any institutional logic. For example, a post on rescuing a software project in the early decades of scientific computing mentions "cold war" and "iron curtain", which we coded as institutions, in this passage:

> The Cold War was frightening: Atomic bombs had been used to bring the second world war to an end, devastating Hiroshima and Nagasaki in Japan. Then there was the arms race. Before the Treaty on the Non-Proliferation of Nuclear Weapons in 1968, all countries were striving to get their bomb (as parodied by Tom Lehrer: "Who's next?").
>
> Conventional military aircraft were widely used on both sides of the Iron Curtain, and radar systems to detect them were deployed to defend national air space. (Blog 1)

On re-reading the coded passages, those concepts didn't appear in connection with the actions performed or decisions taken by any person or organisation in the

rest of the article, therefore they didn't contribute any institutional logic.

Many institutions only appear in relation to other institutions. For example, in this extract:

> In-person learning entails a training or development program conducted at a physical workplace or training center, where employees attend courses, workshops, or training sessions in person, rather than through online or remote means. (Blog 15)

the institutions "physical workspace" and "training centre" are closely associated with "in-person learning", and indeed they only appear in this extract. Therefore we combine the logics of physical workspace, training centre, and in-person learning, and treat them as a group under the name "in-person learning".

We ended up with 127 annotations that relate institutions identified in the blog posts with each other and with activities for people to undertake, and logics that support certain courses of action or legitimise some actions over others. Subsequently, we input the institutes, relationships, and legitimising logics from the annotations into a graph using the `graphviz` tool[5]. To demonstrate this process, figure 7.1 shows a graph constructed from the annotation given in section 7.2.

The complete graph is available digitally as `blog_graph.png` in https://doi.org/10.25446/oxford.24908523. Due to the size of the graph we do not reproduce it here, as it is not legible when shared at a size appropriate for printing. The graph contains over 550 edges, which represent relationships between entities that we coded through the initial coding process and extracted in our annotations.

Some edges are between institutions and actors, in which case they represent logics. Some edges are between institutions and institutions, which represent relative legitimisations. And some edges are between actors and actors, which represent interactions that could represent advocacy, coercion, or mimesis.

To analyse the graph further, we focused on particular aspects of the graph that relate to our research questions. There are significantly more concepts and

[5] https://graphviz.org

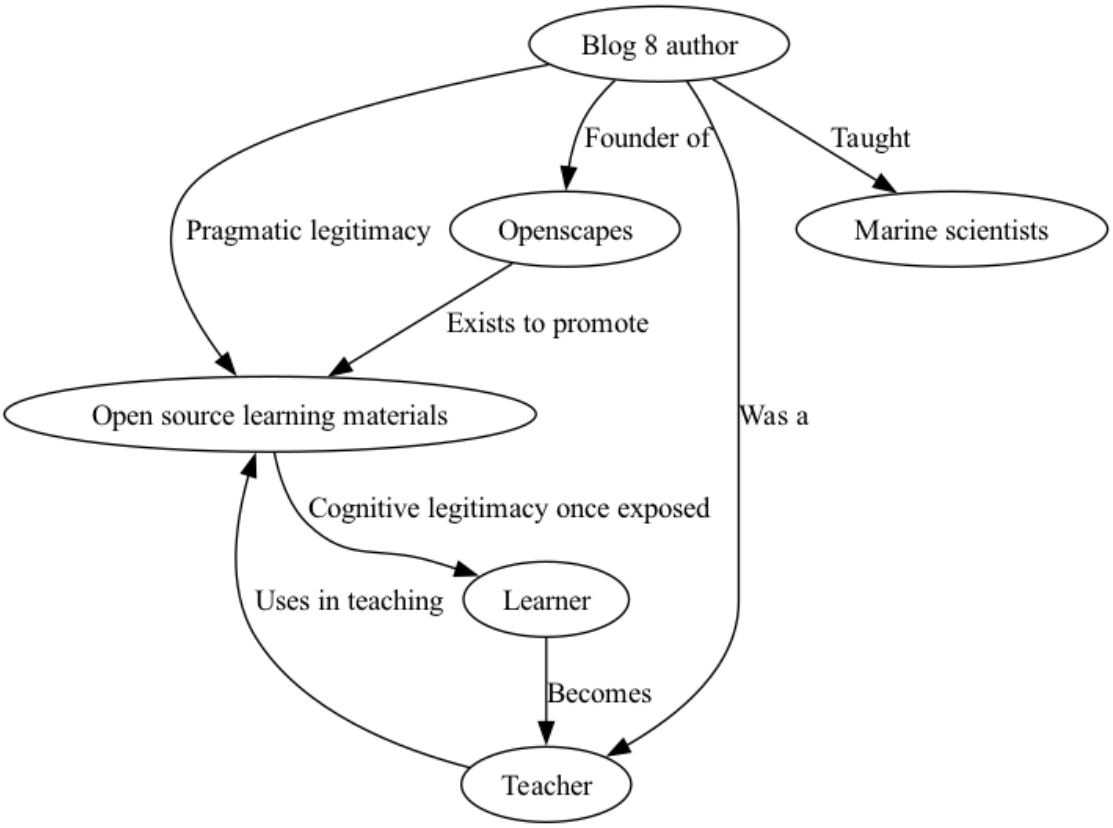


Figure 7.1: A graph representing the institutions and logics identified in the annotation presented in section 7.2. This figure is also available as `example_graph.png` in https://doi.org/10.25446/oxford.24908523.

connections in our data that do not directly relate to the research questions and could be fruitful topics for future investigation.

To investigate RQ3, "What are the different factors and forces a RSE negotiates as they perform their work?", we focused on the parts of the graph that connect directly to research software engineering, and those that relate to skills, practices, and how they are acquired.

7.3.1 Factors Influencing RSE

As described in section 3.6.5, in NIT the institutional logics influence the decisions taken by workers as they weigh up the different logics and their legitimacies to identify which tasks to undertake in any situation. Additionally, performative work including advocacy and asserting normative values means that the relationship

between workers and institutional logics is bidirectionally causal, as workers act to legitimise or delegitimise institutions and their logics.

Therefore, the institutions and logics identified in these blog posts are both factors that influence how RSEs consider and perform their work, and the subject of performative work in which RSEs promote these factors.

In initial coding, we found that RSE was mentioned with a number of different meanings.

- An identity that people adopt, as in:

  > I felt that my development as an RSE would be best done in the context of a local group of peers who I could learn from and potentially collaborate with (Blog 2)

  > RSEs are skilled in software engineering best practices such as version control, testing, and documentation, and they often work in interdisciplinary teams to collaborate with researchers from different domains. (Blog 12)

- A role that people assume, as in:

  > Learning about the RSE role was a bit of a revelation, and gave me a clear sense of the direction my career could take (Blog 2)

  > Another example of research infrastructure roles is the Research Software Engineer, Research Data Scientist and Research Computing Engineer roles. (Blog 4)

  > After his presentation they talked about his thesis and questions such as: is RSE actually a distinct role/discipline and what the future might hold. (Blog 3)

- A career path that people can follow, as in:

  > Specifically, the site's topic on Research Software Engineers delves into the RSE career paths and the RSE community (Blog 12)

> Recognition of research infrastructure roles [including RSE] as compelling career pathways in data science will ensure the quality, integrity and reusability of research outcomes, building a more open, equitable and sustainable future for research. (Blog 4)

- A community of practice, as in:

> He is a 2022 BSSw Fellowship honorable mention, a member of the United States Research Software Engineer Association and ACM, and an IEEE senior member serving in several technical venues. (Blog 11)

> My first goal after attending the online 2021 RSE conference was to connect with any existing local community of RSEs, as I felt that my development as an RSE would be best done in the context of a local group of peers who I could learn from and potentially collaborate with. (Blog 2)

- An activity that people undertake, as in:

> Research software engineering (RSEng) is a field that focuses on the development and application of software tools, techniques, and practices to support scientific research in various domains. (Blog 12)

> Earlier this year, one high-profile Professor advertised for 'RSE Postdocs', including my name on the advert, and offered to allow me to use some of his new recruits' time in providing RSE support within the university. (Blog 2)

- In one case, RSE is used as an abstraction that exists but doesn't convey any specific connection with the people involved, as in the first two instances in the following example.[6]

> While my attempts at networking within my South African university quickly made it clear that the term 'RSE' was effectively unknown here (although I did of course

[6] We interpret "RSEs" in "the support of RSEs" as a reference to RSE-as-identity.

> find 'proto-RSEs' like myself), I also found myself talking with many researchers from different Faculties who—on hearing me explain RSE—were quick to insist that they needed the support of RSEs in their research. (Blog 2)

We found RSE explicitly mentioned in only 5 of the 17 articles we analysed. Additionally, alongside the quotes given above, the title of the blog that Blog 5 appears in is "Research Software Engineering", though the article itself does not contain the term. We hypothesise that as RSE is still an emerging occupation and identity, post authors may choose not to use the term when writing about their field so that readers for whom the content is relevant, but who do not identify with the RSE identity, still find the material applicable to their interests and situations.

For this analysis, we first copied any edge from the original graph that connected to one of the nodes identified above. Then we combined all of these nodes into a single node that represents any aspect of RSE.

We then examined the nodes that connect to the RSE edge, and identified higher-level categorisations that allowed us to group those nodes and build theoretical abstractions. To maintain connection with the source data, we represented the higher-level categories as intermediate nodes that connect to both the RSE node and to the node that we identified as a member of the category. In some cases, sub-levels of categorisation were evident in the data. The resulting graph (available as `rse-focus.png` at https://doi.org/10.25446/oxford.24908523) has a multiple-level star topology, with few connections between distinct categories.

We see that academia is an important aspect of RSE as discussed by bloggers in the field, providing both a context for the occupation and a source of resources that RSEs can take advantage of. One blogger (Blog 2) reports that RSE is "a critical part of the academic ecosystem", which appeals to the existing legitimacy of academia to promote the moral legitimacy of the RSE activity: if one considers academia to be important, then they must also value RSE.

The same article points to the importance of academia as a source of funding for RSE. The author describes a "chicken-and-egg" funding problem for RSE: the culture

must support writing RSE work into academic grant applications, but legitimising that support requires evidence of success.

Academia appears to enjoy a cognitive legitimacy among the articles investigated. While government institutions, including the Alan Turing Institute in the UK:

> As the UK's national institute for data science and artificial intelligence, the Alan Turing Institute's mission is to make great leaps in data science and artificial intelligence research in order to change the world for the better. (Blog 4)

and the Department of Energy:

> The webinar series is produced by the IDEAS Productivity Project in collaboration with the DOE/ASCR computing facilities (ALCF, NERSC, and OLCF) and the Exascale Computing Project. (Blog 9)

and others in the US are mentioned by various authors, no other source of funding or structure for employing and organising RSEs is discussed. This could be due to a lack of data, or an implicit assumption on the part of the authors that RSE is a part of the government-funded research landscape that comprises universities and state-run institutes, leading to the isomorphism with university structures that sees "RSEgroups" seeking recognition in the university, appealing to traditional university decision-makers including professors and, in Blog 2, the "Deputy Vice-Chancellor for Research and Innovation".

To negotiate the work that they do and its place within a larger system, RSEs collaborate with others. We identify two categories of collaborators: the academic roles—scientist and researcher—and the "RSE-like" roles: the peer network of other RSEs, leaders of RSE groups, and "proto-RSEs".

Considering the example earlier in this section, a "proto-RSE" is somebody at a university who supports research in an RSE-like way but doesn't identify as an RSE. We could consider proto-RSEs to fit both categories of collaborators, as RSE-like people who occupy existing non-RSE roles at universities. Proto-RSE could act as a precursor to mimetic isomorphism, if an institution identifies its proto-RSEs and

labels them "RSE" then it can be seen to be performing RSE. The presence of proto-RSEs is used in the blogs to lend cognitive legitimacy to the institution of RSE: the activity is already happening, even if it is not yet labelled and explicitly supported.

RSE group leaders are identified by the author of Blog 2 as being sources of information on how "RSE groups are founded and run", a topic which they discussed with the group leaders at a conference for RSEs. Unfortunately neither their article nor any other we considered defines the activities of a group leader that enact founding and running an RSE group. However, the author implies a process that is both mimetically isomorphic (not knowing how to do it, they look to examples from those who have already done it in other institutions) and normatively isomorphic (the RSEs and their group leaders share a culture through their conference).

Researchers make use of "scalable coding support" that RSEs provide (Blog 2). The activity of "research", particularly "computing research", is said to require:

> [a] culture of reproducibility and replicability in computing research needs attention and focus for ensuring transparency, rigor, and trust in scientific research and its results. (Blog 6)

These attributes are provided by tools that RSE employs, for example:

"transparent and reproducible workflows that can be shared openly on GitHub." (Blog 8)

Here we see—albeit spread across multiple articles in the source data—a single, deductive argument that supports the application of RSE to research. If we accept the normative axiom that computing research needs reproducibility to increase transparency, then the fact that RSE supports transparent and reproducible workflows means that computing research can take advantage of RSE. Therefore RSE has both normative legitimacy (by supporting the value of transparency) and pragmatic legitimacy (researchers performing computing research can look to RSE to get the transparency they value).

7.3.2 Skills and Practices of Research Software Engineering

We then took a different approach to understanding the factors influencing an RSE's work, the subject of RQ3, and looked for skills and practices described in the blogs, and for logics related to skills acquisition. We reason abductively that these represent logic that the authors implicitly associate with the institutions of RSE, as they introduce the logic in posts that publicly perform a definition of the field. Therefore we find relevant skills and practices even in situations where the author does not mention RSE explicitly, which was the case in 12 of the 17 articles analysed. An author may use the passive voice or otherwise avoid identifying an actor in their blog post, as is the case with this example:

> This topic of online learning includes materials that place notable focus on software development, efficiency, and sustainability, as distinct entities from the "scientific" and mathematical aspects of computational science and engineering. (Blog 14)

We can see in this sentence that online training supports particular topics, but not who should understand those topics or take the training. However, given the performative nature of blogging as described by Powell and Colyvas [215], and our focus in searching for blogs on the topic of RSE, we can infer that these skills and training outcomes are mentioned because the author is including them in their contribution toward the reality of RSE that they help to define.

As we started our analysis with a category of ideas rather than with a central node as in section 7.3.1, the topology of the resulting graph (available digitally as `skills-focus.png` at https://doi.org/10.25446/oxford.24908523) is very different. The graph has a more complex structure than the star uncovered in that section, with more connections between nodes at different levels.

Software Engineering Skills

The majority of practices identified are aspects of software development. We identify a practice as relating to RSE where an author explicitly mentions that the practice is something RSEs do, or where they advocate for its consideration or

adoption. To learn more about the breadth of software development practices undertaken and the connections with software engineering, we compared the collection of software development practices identified in the blogs data with the IEEE Software Engineering Body of Knowledge (SWEBOK). If we connect the practices in the blogs with the knowledge areas (KAs) named in the SWEBOK [216], we find the following software engineering KAs are mentioned in the blog data.

- Software construction. For example:

  > This topic of online learning includes materials that place notable focus on software development, efficiency, and sustainability, as distinct entities from the "scientific" and mathematical aspects of computational science and engineering. (Blog 14)

- Software testing. For example:

  > RSEs are skilled in software engineering best practices such as version control, testing, and documentation, and they often work in interdisciplinary teams to collaborate with researchers from different domains. (Blog 12)

- Software configuration management. See the previous example, in which "version control" is an example of an approach to software configuration management.

- Software engineering management. For example:

  > The article highlights several common mistakes in project management that can have a detrimental impact on software development. These mistakes include prioritizing planning and documentation over communication and collaboration, focusing on deadlines and budgets at the expense of quality and user satisfaction, assigning tasks without considering team members' skills and interests, and micromanaging team members, among others. (Blog 13)

- Software quality. See the previous example.

- Software engineering professional practice. The example of software testing given above also mentions "documentation" as a best practice, which is part of the Communication Skills sub-area of this KA.

- Software engineering economics. For example:

  > Hence, Julia fills a gap at the intersection of high performance and high productivity for scientific software. (Blog 11)

- Computing foundations. The above example is an extract from a post on the Julia programming language that compares it with other tools: Programming Language Basics is a sub-area of this KA.

Conversely, these KAs are not mentioned in the blog data:

- Software requirements.
- Software engineering process.
- Software engineering models and methods.
- Engineering foundations.

While these KAs are mentioned once, in a list of activities incorporated in RSE in Blog 12, but not explored in detail:

- Software design.
- Software maintenance.

Their absence suggests that the institution of software engineering (including the IEEE Computer Society, who produce [216]), does not have strong legitimacy among RSEs. If software engineering were the dominant logic in RSE, we would expect to see a closer association between the field's knowledge areas, and the expected behaviour of RSEs. It is also possible that these KA s are missing from our data because we did not discover the blog posts that include them, or because these areas are considered

"solved" in RSE and therefore uninteresting to write about. We connect the findings from this chapter with our other data in chapter 9 to disambiguate these possibilities.

Software engineering practices that are present include those that support solutions in the research domain, high-performance computing (HPC), scientific computing, and the Julia programming language; and general software engineering practices (project management, productivity, documentation, testing, version control, coding, sustainability, and efficiency).

Skills Relating to Open Source Software

In section 3.2.3 we saw that many authors connect the values of OSS to those of RSE. Four of the blog posts we analysed referred to open source software, including two (Blog 8 and Blog 16) in which the post titles make it clear that open source software is a central topic of the content. Mapping the content in these posts to the OSS Watch software sustainability maturity model in [8] (SSMM), we find that the topics in these articles cover a wide range of maturity levels, from Blog 8 which discusses building training programmes from material that is published in open-source repositories (SSMM level 2), to Blog 16 which describes guidance for attracting and developing contributors, addressing funding needs, and the project roadmap is defined and accepted within the software's community (levels 4–8).

Other Research Software Skills

As we discussed in section 2.4.3, various authors identify skills that research software engineers need beyond the general software engineering practices in SWEBOK. Considering Goth et al.'s list of foundational RSE competencies [88, §4], with "4.1.1 Classical software engineering skills" equated by the authors to the SWEBOK and investigated above, we find that the following competencies are discussed in the blog posts we analysed:

- 4.1.3 Creating documented code building blocks (3 posts)
- 4.1.4 Building distributable software (3 posts)

- 4.1.5 Use software repositories (2 posts)
- 4.2.3 Software re-use (4 posts)
- 4.3.2 Teaching (3 posts)
- 4.3.3 Project management (2 posts)

Meanwhile, the following competencies are not mentioned in any of the posts:

- 4.1.2 Adapting to the software life cycle
- 4.1.6 Software behaviour awareness and analysis
- 4.2.1 Conducting and leading research
- 4.2.2 Understanding the research cycle
- 4.2.4 Software publication and citation
- 4.2.5 Using domain repositories/directories
- 4.3.1 Working in a team
- 4.3.4 Interaction with users and other stakeholders

It is likely that those competencies that are not specific to RSEs would not be considered "on topic" for a blog on research software engineering: for example, items 4.2.1 and 4.2.2 are general research skills and as such might be assumed to be background knowledge by authors talking to RSEs. However, other competencies—for example, items 4.1.2 and 4.2.4—are clearly in the domain of RSE.

Skills Related to Specific Types of Research Software

In section 2.2.1 we listed examples of typologies that classify research software by its position in a "stack" [5], or by role [6]. We don't see evidence of these categorisations (or others) used in the blogs; instead, "research software" is typically referred to as a whole.

This could indicate that RSEs are not thinking about these typologies when writing about their work. Alternatively, that the blog authors assume their audiences are working on particular categories of software, and that the position in the typology is implicit. For example, there could be a tacit assumption of a division of labour in which RSEs concentrate on reusable components in the lower levels of Hinsen's stack, and "end researchers" combine and integrate these components in scripts for particular experiments.

Scientific Skills

The remaining practices we identified are scientific in nature, as in this example where science and mathematics are described as "aspects" of computational science alongside other software skills:

> This topic of online learning includes materials that place notable focus on software development, efficiency, and sustainability, as distinct entities from the "scientific" and mathematical aspects of computational science and engineering. (Blog 14)

or relate to replicability and reproducibility, normative values for scientific research:

> Advancing the culture of reproducibility and replicability in computing research needs attention and focus for ensuring transparency, rigor, and trust in scientific research and its results. (Blog 6)

While it is not surprising to find that RSEs are expected to provide skills that support the needs of science, given the connection to research and dependence on institutions like universities and government research agencies, it is interesting to

note that other areas of practice for research—social sciences, arts and humanities—are absent from the data. This bias towards science could be environmental (perhaps there is more science being done than other branches of research in places where RSEs work, and therefore RSEs are more likely to encounter and work with scientists), or it could be selective (perhaps people working in scientific fields are more likely to identify as RSEs, or to encounter RSE in their research work, or to blog about the field).

Either way, there is a tacit expectation that the Research in Research Software Engineering is scientific research. In section 3.2 we noted that the Arts and Humanities Research Council discovered an unmet need for research software skills [106], so we reject the possibility that RSE is a scientific discipline because only scientific research requires RSE support. The field may not serve the needs of other research disciplines, or RSEs working in those disciplines may adopt scientific computing practices as they face a lack of domain-specific support. As we saw in chapter 5, people working in non-scientific faculties were under-represented in our survey.

7.3.3 Routes to Skills Acquisition

Many of the practices identified in section 7.3.2 are not connected with learning modes in the blog data. In other words, we understand that RSEs may be expected to perform these practices and be skilled in their application, but we don't learn where they acquire the skills.

The venues that are explicitly linked to RSE practices include:

- conferences, where one can learn about HPC:

  > The ISC High-Performance conference series is one of the premier venues for high-performance computing and computational science and engineering. (Blog 9)

  and the Julia programming language:

> JuliaCon is the annual community gathering; a variety of interesting talks and tutorials from there can be found on YouTube." (Blog 11)

- workshops, sharing information on the Julia programming language.[7]

> Last summer we organized a full-day workshop, entitled Julia for Oak Ridge National Laboratory Science, JuFOS, which, to our surprise attracted 101 registrations from a range of scientific domains. (Blog 11)

- online learning resources covering HPC, programming, data analysis, sustainability, efficiency, and "software development" broadly:

> Universities and specialized online learning companies provide formal, structured and comprehensive courses that cover a broad spectrum of topics, from fundamental programming and data analysis to advanced parallel programming. [...] This topic of online learning includes materials that place notable focus on software development, efficiency, and sustainability, as distinct entities from the "scientific" and mathematical aspects of computational science and engineering. (Blog 14)

- universities (see the quote on online learning resources, above);
- and an ACM emerging interest group on reproducibility and replicability:

> The Association for Computing Machinery (ACM) Emerging Interest Group on Reproducibility and Replicability aims to promote and improve research reproducibility in the computing field by establishing best practices, offering education and training, and providing resources and tools to enhance transparency and replicability. (Blog 6)

Additionally, two other specific training organisations are mentioned: Software Carpentry in Blog 2 and Openscapes in Blog 8. We infer that these organisations are mentioned because they are of interest to RSEs, and therefore that the topics

[7] Another article, Blog 4, mentioned the "Collaborations Workshop '23" but did not describe its scope.

they provide training on are topics that the authors who mention them expect RSEs to learn.

When accessed on 6th December 2023, the Software Carpentry website [217] offered online learning material on these topics:

- The Unix Shell
- Version Control with Git
- Programming with Python
- Plotting and Programming in Python
- Programming with R
- R for Reproducible Scientific Analysis

Connecting these topics with categories identified above, we see programming language fundamentals, software configuration management, and reproducibility. Additionally the new topic of "plotting" (in the sense of graphical visualisation of data) is introduced; this topic bridges both the scientific and software domains, as graphing is a common practice for visualising data in scientific research, and is a form of user interface which is a field within the "software design" knowledge area in the SWEBOK.

The Openscapes website [218] accessed on the same date offered online learning material on these topics:

- R for Excel users
- Making sharable documents with Quarto
- Transitioning my workflow to the cloud

These topics relate to programming language fundamentals, software engineering professional practice, and to the distributed computing section of the SWEBOK "computing foundations" knowledge area.

We still find that the coverage of both software engineering and scientific knowledge expected of RSEs is incomplete, so that the blogs that describe pathways to skills acquisition omit explanation of how many practices are transmitted.

7.4 Discussion

7.4.1 Summary of Findings

We searched for evidence of people blogging about the occupation of RSE, and gathered data from posts published in early 2023 to form an analysis of the public performance of research software engineering. Our discovery phase yielded 17 articles on diverse topics and by diverse authors, which we initially coded on a sentence-by-sentence basis.

We then applied NIT to categorise the coding based on evidence of institutions, logics, and their legitimising factors.

To interpret these data, we organised them graphically and inspected the graph, preparing subgraphs focussed around particular topics of interest.

We discovered that RSE depends strongly on the logics of research, with the same institutions and state-supplied government sources that employ and fund researchers and their activities also employing and funding RSE.

Additionally, we found that the practices of software engineering are incompletely adopted in RSE, with six of the 14 knowledge areas described in SWEBOK completely missing and many of the remaining 8 only partially present. Other skills that are described as foundational competencies for RSE in the literature are also covered to varying levels in the blogs, with eight of the 15 competencies listed in [88] completely missing. The skills and practices that RSEs do demonstrate are primarily disseminated through peer venues like conferences, workshops, community-provided online resources, and special interest groups, where their dissemination is explicit.

7.4.2 Implications of Findings

In the previous chapter, we concluded that RSE occupies a liminal space in the shadows of academia, and that practitioners saw difficulty where the supposedly novel goals of RSE weren't met by distinct expectations from academics and their institutions. The blog data, and our analysis using NIT, allow us to begin to explain this apparent dichotomy.

The funding that supports RSE is the same funding that supports traditional research activities, and the people come from the same backgrounds and work in the same institutions as other research roles. In NIT, these properties are predictors of isomorphism, in which RSE would be expected to adopt the institutions and logics of academia, which is what we see and consistent with the difficulties described in the interviews.

We theorise that the absence from the blogs of several knowledge areas described in the literature is due in part to RSE's strong isomorphism to research, and the fact that these practices are not expected in the "parent" research fields; and in part to mimetic borrowing of software engineering knowledge focusing on highly-visible areas first.

We considered these possible explanations for the omissions of some skills and knowledge areas from RSE blogs:

1. The sample of blog posts considered is small, so that theoretical saturation is not reached. We would find other examples by including more data.

2. Practitioners know that they are only employing a limited range of software engineering practices, but have prioritised these as the most relevant to research software.

3. The absent skills are passed on through commonly-known routes that the authors did not find necessary to explicate in their blog posts. For example, perhaps "everybody knows" that scientific knowledge is acquired through university degrees.

4. The absent skills are tacitly assumed and absorbed through practice or self-learning: there isn't an explicit pathway to bring attention to.

5. The absent skills represent a real education gap: the skills are needed but not formally taught.

We will discuss the first of these explanations, that we have not gathered enough data to exhaustively enumerate software engineering practices in RSE, in section 7.4.4, below.

We consider the second explanation, that RSEs collectively identified and prioritised a small number of software engineering practices for "early adoption" in research, problematic because the data do not contain references to those priorities or to a prioritisation effort. There are some indications in the research software literature that scholars consider the priority of skills adoption; Wilson et al. write:

> [A]s computing has become an essential part of science for all researchers, there is a larger group of people new to scientific computing, and the question then becomes, "where to start?" This paper focuses on these first accessible skills and perspectives—the "good enough" practices—for scientific computing: a minimum set of tools and techniques that we believe every researcher can and should consider adopting. ([114])

And this author—with colleagues—wrote of "a choice of practices was designed to be small enough that every researcher can either acquire the relevant skills or find someone locally to provide training, yet impactful enough to bring about a meaningful improvement to the state of research software." [115].

Taking this information from the literature into account, we could hypothesise that a small number of software engineers have considered which practices are important and shared their suggestions in scholarly journals; and that in the performance of RSE these lists are taken "as read" and not critiqued as part of the everyday practice.

We theorise that the remaining three explanations are correlated and explain the absence of explicit learning pathways for many of the expected practices in RSE.

We theorise that as RSE is an emerging discipline, specifically emerging from academia, the logic of skills acquisition in academia is applied: in which the last formal education point is the higher degree, and people are subsequently expected to be self-sufficient and independent learners who pick up knowledge from their own study, and from networking at conferences and seminars. Supporting this hypothesis is the observation that universities are only mentioned once as learning venues, as "Universities and specialized online learning companies"; as many RSEs work in universities we deduce that university learning is a tacitly-known background to their activities.

Additionally, RSEs are entering the field with academic backgrounds and from various points in an academic career, so have picked up certain skills along that path. This means that people entering the field of RSE have those skills, and those around them (the other researchers in their original field, and the RSEs working with their research groups) also have those skills, making them likely to be tacitly assumed. If the research groups are able to sustain a level of success without acquiring other skills, then the benefits of acquiring those skills can be missed. This leads to an ironic position in which RSE brings value because it focuses on software, but the level of focus on software and the skills needed to work on the software are tacitly equivalent to those available without RSE.

Supporting the explanation that the software skills are unknown in the research domain—even among RSEs—and represent a real education gap, is Schönborn's 2023 survey of mathematics and physics researchers who moved into careers as software engineering consultants. He surveyed the consultants on the usefulness to scientific computing of various software engineering concepts in the field of cloud-native computing, using a four-point Likert scale with "I don't know" as an alternative response, and found that for a majority of concepts studied, "I don't know" was the most frequent answer [219]. Examples of software engineering concepts for which the most popular answer was "I don't know" include containerization, microservices, and infrastructure-as-code. Only for continuous integration ("very useful") and agile development ("useful") was the most frequent response committal.

This hypothesis also supports an explanation for the partial borrowing of software engineering practices described earlier in this section. If RSE is isomorphic with the research that hosts it, and the SWEBOK is not well-established already within research, then increasing its legitimacy is a very difficult task.

We propose that RSE is isomorphic with academia with respect to professional development for normative reasons, consistent with hypotheses A-1 and B-1 from [188] described in section 7.1.

The observation that research software categorisation or typology does not appear in the blog data is also consistent with the proposition that RSE is isomorphic to academia. Participants in the research-producing ecosystem create or reuse whatever software is necessary to achieve the research goals of securing funding and creating publications. For a group to set a goal of creating a reusable component at a lower level of the Hinsen "stack" [5] would require a plan that expends resources on community-building and supporting other researchers publishing work that this group does not necessarily receive academic credit for, which would be counter-intuitive given academic incentive structures.

This gives us the outline of an answer to RQ3: the factors that influence an RSE's work are primarily shaped by a need to thrive in, and remain comprehensible to, the academic ecosystem; and secondarily the knowledge that software expertise can play a part in that navigation of academia, to the extent that the software practices themselves can be safely introduced into the logics of research.

Taken together, the interviews analysed in chapter 6 and the blogs considered in this chapter create an image of RSE as a small collection of change agents trying to bring academia around to a new way of thinking, and struggling to overcome inertia. This motivates finding an answer for our remaining research question, RQ4: how does RSE change the way research is done? It seems unlikely that RSE with its strong isomorphism with the institutions and logics of research will become independent, so we look for ways in which it observably affects research to understand the extent to which RSE can "bring research along for the ride" and enact the changes its practitioners and proponents envisage.

These findings are consistent with the "liminal occupation" in the shadows of academia that we uncovered in our interviews and discussed in section 6.4. In section 9.3 we undertake a deeper comparison between the findings of this chapter and our other data.

7.4.3 Methodological Contributions

We find that blog data represent an interesting lens into how practitioners of an occupation publicise, evangelise, and justify their work to a broad audience. Other researchers of occupations would find blogs to be an informative data source that demonstrate what topics are of interest—and suitable for public discussion—within their domains of study. Particularly, researchers whose work focuses on academic fields should see blogs not as "grey literature" that is adjacent to scholarly publishing, but as a distinct activity and data source with its own norms and motivations.

As a contrast to the other sources of data we used, the researcher takes a more passive role in analysing blog data than in interviews where they set the questions and engage in discussion with the participants. In this way, incorporating blogs as sources of data acts as a useful test of the reflexivity of a researcher's analysis when their results are compared to those gathered through more active interventions.

Neo-institutional theory provides a powerful framing for analysis of these data, as it allows for activities that are ceremonially, rather than rationally, adopted and provides explanation for practices that are adopted "because that is what we (or they) have always done" rather than because of economic benefit. In the context of a constructivist methodology, NIT is a good fit as it allows the relationships between different actors, and between state and non-state institutions, to arise from the data and its analysis. This contrasts with other theories of professions, including Marxian and Weberian approaches, where the relationship between the state and other institutions is assumed *a priori*.

7.4.4 Threats to Validity

As with chapter 6, we do not believe that we achieved theoretical saturation in analysing the 17 articles described in Appendix B. The topics discussed in these articles represent a small chronological window (between January 2023—April 2023) into blogging relevant to RSE, filtered by searching for particular terms that we constructed *a priori*. They may therefore represent a snapshot—or a partial snapshot—of the "zeitgeist" among RSEs, and topics discussed in the articles examined could have been important at that time only. Conversely, other topics that were important in RSE blogs at other times may be unrepresented in our data.

We also consider that blogs are a performative form of discourse [215], and that bloggers may avoid writing about certain issues to avoid making them part of the reality that the blogs are constructing, or drawing attention to them in a public context.

7.4.5 Further Work

We acknowledge that this work could be expanded to achieve theoretical saturation, and consider two pathways to achieve that. Firstly, the scope of blog posts included in the study can be expanded, by increasing the date range, or by widening the inclusion example (for example, by including websites without an RSS feed). Secondly, other sources of data that represent performative discourse in the RSE field can be included. We note that our own data includes references to podcasts (Blog 3), workshops (Blog 4, Blog 11, and Blog 10) and other venues where RSE is performed; these could be relevant sources of similar data, albeit with different motivations and forms of presentation.

The work presented in this chapter raised a number of tentative hypotheses that warrant further examination. These are:

- People who are RSEs, or who are writing for RSEs, may avoid using the word to avoid restricting the potential audience for their writing.

- The funding landscape for RSEs is coterminous with state-supported funding for other research activities.
- The institutions of software engineering, for example the IEEE Computer Society, do not enjoy a strong legitimacy among RSEs.
- The "Research" in RSE is predominantly *scientific* research, and RSE doesn't provide strong support to other research disciplines.
- There is no ongoing practice of critically engaging with the SWEBOK to decide which software engineering practices are of highest priority in RSE.
- Professional development in RSE is strongly isomorphic with academic development and relies strongly on self-education.

Further, we note that we coded an identitarian definition of RSE (i.e., someone *is* an RSE) that was distinct from the practice, or the occupation, of working with software in research; labeled by one author as "proto-RSEs". We consider it of practical importance to those in leadership positions in RSE to explore this distinction, both in quality (what is it that makes someone "an RSE" rather than a researcher with a software interest) and in magnitude (how many people who aren't RSEs but who perform the same work are there) to inform outreach and legitimisation activities.

7.5 Conclusion

Despite having "software engineering" in the title, we discovered that "research software engineering" only covers a fraction of the software engineering body of knowledge. We applied NIT to an analysis of public blogs about RSE to theorise that multiple properties of the field—the institutional context in academia, the reliance on state funding through research councils, and the normative background of many participants in research—lead to isomorphic tendencies in RSE that make the field more like research, and less like software engineering.

We further theorise that where RSE has adopted practices and skills from software engineering, it is done mimetically. The institutions of software engineering do not have the legitimacy within RSE for their logic to dominate against the prevailing research isomorphism.

We found that the results of this analysis are consistent with the results in the previous chapter, in which RSE was found to occupy a liminal space in the shadows of academia, and that the skills and practices RSEs employed were often self-taught.

Finding that RSE is a partial application of software engineering ideas to research, with a goal of changing how research engages with software, motivates us to explore RQ4: how does RSE change the way research is done? In the next chapter, we will seek an answer to that question, through focus groups conducted with research groups that create research software.

8

"I'll Get You a Pint if You Find a Bug"—Research Software Communities of Practice

Contents

8.1 Introduction

In chapter 6, we found that RSE is a liminal role on the edge of academia. In chapter 7, we discovered that this liminality is associated with a dependence on the institutions and logics of academia, and that the adoption of software engineering practices within RSE is incomplete.

We now begin the early steps in an exploration of the relationship RSEs have with each other and the researchers with whom they work, to understand more about the context of the "software engineering" within "research software engineering". Seeing how people who work on research software collaborate with their peers, colleagues, and broader communities can help us to understand how individuals negotiate the competing logics of academia and software engineering, and build an answer to RQ4 by exploring working relationships in different organisational contexts. We use the theory of Communities of Practice to investigate these relationships.

8.1.1 Undertaking Focus Groups

Focus groups add another dimension to the data we collect in this study. Surveys and interviews are both undertaken individually by a single participant at a time, with the relationship between the participant and researcher in the foreground in interviews and in the background—though not quite nonexistent—in a survey. The relationship between researcher and "participant" is also minimised when we research blog posts, with the researcher selecting and interpreting content that the authors have already prepared. We might often assume that a blog post has a single named author and that the medium represents a one-to-many communication pattern, but should remember that the authorship and credit framework for blogs is much more informal than for peer-reviewed literature and that a single post may be the work of multiple contributors, even where it only identifies one creator.

In a focus group, the researcher facilitates conversations between participants, encouraging them to discuss topics among themselves rather than responding to the facilitator when prompted. This is a very different dynamic than an interview, meaning that while the data reflects a certain amount of direction from the facilitator, we mostly record interactions between the participants and conversation that they take in a collectively-defined direction.

Additionally because the participants are having discussions with colleagues and people who they might be in reporting or management relationships with, we observe interactions that include social and political dimensions that are absent both from one-on-one interactions with interviewers, and from blog posts that authors write for an abstract, public audience.

8.1.2 Communities of Practice

The theory of communities of practice, which we introduced in section 3.6.6, is well-suited to analysing data from focus group sessions to discover an answer to our research question RQ4 on the relationship between organisational structure and software engineering practice. Communities of practice form when people who share a passion for an activity interact to improve at performing that activity.

By analysing the way in which participants in the focus groups interact with each other and what they say about the practices they perform at work, we can uncover the extent to which practices are communicated through formal structures or through informal communities of practice. We can also discover whether participants form or join communities of practice that involve people outside their immediate research group, for example collaborators from other research groups or members of RSE associations.

8.1.3 Initial Evidence

The data we gathered for 6 and 7 already gives some insight into how RSEs engage with communities of practice.

Interview participant 4 described the Research Data Alliance as a "grass roots, ground-up organisation... who work with research data and software and come together to come up with principles and best practice guidance on how to do things like FAIR data." This formulation neatly includes the three dimensions of practice that Wenger defines as the properties of a community: mutual engagement; a joint enterprise; and a shared repertoire [190, pp.72–73]. The participant indicates that community norms in research drive the practices accepted by research institutes and funders: "the data authors were also saying things like, oh the Research Institute or the research funders should make us do this [... but] we are guided by our research communities as to what we say is mandatory and what isn't."

Participant 7 described the benefits of the shared practices in the RSE community as lending credence to those practices when advocating for them in their work, and giving them a more concrete sense of the legitimacy of their position on that work so that they "felt more convinced that I should hold this opinion and not be swayed."

As discussed in section 7.3.3, bloggers writing about RSE discuss many community-based training opportunities including conferences, workshops, and both in-person and online learning scenarios. As such, RSE represents a community in which ideas and practices are shared, promoted, and normalised.

In this chapter, we study the day-to-day work of particular research groups to understand how these abstract ideas of grass-roots principles and community-based education manifest in the lived realities of researchers and RSEs working on research software.

8.2 Method

We invited research groups to participate in one-hour focus groups, held in meeting rooms at the participants' institutions and facilitated by the author. We sent the following recruitment message to potential participants, via the society of RSE Slack online chat service, the Oxford University Research Software Developers' Network Slack service, and individual email to the principal investigators of groups that we and our supervisors knew worked with research software:

> Dear researcher,
>
> My name's Graham Lee, and I'm studying how researchers use and create research software for my D.Phil in the Department of Computer Science, at the University of Oxford, supervised by Professor David Gavaghan. I'm approaching you because I believe that your group works with research software, and I'd like to invite you all to a focus group, to be hosted in person in your institution, about how you work together to create research software.
>
> If you agree, we would find a date and time that works for your team to come into a meeting room, and spend about an hour on activities and discussions exploring how you use and write software in your research. I would video-record this meeting, anonymise your contributions, then analyse the contributions for inclusion in my thesis, to be submitted in January 2025. You will be able to withdraw your participation at any time before 31st October 2024. More information, including my research question and the research ethics information related to this study, is available in the attached information sheet. Please contact me by email graham.lee@cs.ox.ac.uk if you have any questions, or to take part.
>
> Kind regards,
>
> Graham.

The information sheet described in the message is included in appendix D. The participants of each focus group were members of a single research group, so that the existing relationships and shared experiences the participants had created with their colleagues could be brought into the focus groups.

Each focus group lasted for an hour, and was facilitated by the author of this thesis. After confirming that the participants had received the information about the research ethics of the study and inviting them to sign a consent form, I initiated conversation by introducing two activities, and then invited the participants to complete and discuss the activities with minimal further prompting. In this way, I concentrated the conversations on the participants' relationships with each other, avoiding a "round-table interview" scenario where each participant awaited prompting from the researcher before speaking.

The first activity examined how the members of the research group collaborate to achieve their research goals. I asked the group to collaboratively draw a diagram, depicting how they work together and the tasks they perform. I offered that they can draw the diagram however they want: as a timeline; as a Venn diagram where tasks that people share appear in overlapping regions; or any other way that makes sense. In the description of the activity, I suggest that people discuss additions to the diagram before writing/drawing, to socialise and check their suggestions and initiate conversation. I also mentioned "creating or using research software" in the activity prompt, to try to get the participants to include research software activities in their diagram.

The focus group was given ten minutes to create the diagram, then discussed it for twenty minutes, which can lead to the group amending or extending the diagram (sometimes verbally, rather than on the whiteboard). Finally, I took a photograph of the end state of the whiteboard.

Due to the anonymity guarantees made to focus group participants we do not share the whiteboard images here, but used them to clarify aspects of the discussion while transcribing. We also noted in the transcriptions which participants wrote or drew on the board during the course of the activity.

The second activity examined how the research group produces research software, and their perception of their software engineering capabilities. As facilitator, I introduced the Software Engineering Method and Techniques (SEMAT) methodology-agnostic approach to evaluating software endeavours [44, pp. 28–29] to the group. Each member was given a set of cards representing the seven "essence alphas" in SEMAT: Work, Way of Working, Team, Opportunity, Stakeholders, Software System, and Requirements. These cards are provided under the Creative Commons attribution licence by SEMAT, Inc., and are reproduced in Appendix E.

I asked the groups to rank the cards in order of importance to their own software activities, thinking about which ones are most critical to do correctly to succeed at producing research software. I also asked the groups to discuss how well they do at managing each of the areas represented by the essence alphas, and which

Figure 8.1: A photograph of a meeting room set up for a focus group activity.

ones they think important for their team to improve in. The groups were asked to work on the activity for ten minutes then present their ranking and reasoning to me, after which I asked follow-up questions to explore details of their choices. Finally, I took photographs of each group's ranked card stack, if they had laid the cards out during the activity (in one group, participants picked up the cards and discussed their order without physically organising them).

These activities and discussions were video-recorded, with one camera taking in a wide view of the meeting room and one trained on the whiteboard. An omnidirectional microphone in the centre of the room captured audio. The video and audio sources were multiplexed using the OBS application.[1] I also collected field notes during the sessions. The layout of a focus group room—before the participants arrived—is shown in Figure 8.1.

[1]Open Broadcaster Software: https://obsproject.com.

We transcribed the audio recording, referring to the field notes for guidance, and anonymised individual, group, and organisation names using numeric identifiers. Our transcripts include the discussion and non-verbal utterances from the participants and the author, who facilitated the sessions, and gestures made by participants if they were in front of the camera (for example, pointing at other people, shrugging, drawing on the whiteboard, moving SEMAT cards). These gestures are indicated in the transcripts with asterisks (*) surrounding the actions. If people talked over each other, we indicate that with square brackets in the transcript on consecutive lines.

We numbered the lines to simplify referring to sections of the transcripts. The line numbers are given in the extracts below, and correspond to the line numbers in the original transcripts. Due to the redaction of sensitive information and typographical considerations, the lines in the extracts occasionally have additional line breaks, but the line number is not changed at such a break.

To analyse the data, we reviewed the transcripts for discussion that relates to RQ4, and for discussion that confirms or contradicts conclusions made in chapters 6 and 7. We identified the relevant sections and re-watched the video recordings alongside the transcript, to understand the sense of the discussion (for example, whether speakers were talking confidently or tentatively, whether other participants seemed to agree or disagree, and whether participants looked to others for support or confirmation). Then we extracted the relevant excerpt of the transcript into a new document, which we loaded into NVivo for coding. We used the Communities of Practice theory (see section 3.6.6) to guide our analysis and coding, and additionally used codes generated in our prior analyses of interview and blog data.

8.2.1 Description of Participating Groups

In total, we held 3 focus groups. They were:[2]

FG1 Members of a group in a mathematical sciences department who research a field in computational biology, and have commercial applications of their research outputs. This group had three participants, one D.Phil. student and

[2]One of my supervisors, Prof. David Gavaghan, participated in one of the focus groups.

two postdocs, none of whom were the two people in the group who held the role RSE but all of whom write software in support of their research. One of the participants arrived after the focus group had already started, during the first activity. All of the participants in this group were based in the same university.

FG2 Members of a collaboration between multiple physical sciences and research software engineering groups in different institutions, who produce modelling software used in both research and industry. This group had two participants, an RSE group lead and a lecturer, from different institutions.

FG3 Members of a collaboration between multiple science and research software engineering groups in different institutions, who produce modelling software used in medical and biological science research. This group had three participants, two co-investigators from a computer science department and a RSE, who works for an RSE group outside that department but in the same institution.

Each of these groups comprised members who work in research-intensive, pre-1992 universities that are in the Russell Group[3].

The reason for the small number of participants—referring both to the number of focus groups and the number of participants in each group—is that we found it difficult to coordinate availability for multiple people in the same group, so that it took a long time even to hold the three groups described here. We received a low rate of responses to the recruitment message described in section 8.2, and some of those respondents thought they were being asked to participate as individuals. When we clarified that we wished to engage their whole research group, they either declined to participate or were unable to recruit other members of their groups.

Even in groups with broader interest in participation, scheduling time when multiple members could attend an in-person session was difficult and took a long

[3] https://russellgroup.ac.uk

communication chain, often to conclude that it wouldn't be possible. This was particularly a problem in collaborations that comprise members of multiple institutions.

We appreciate that this is a small number of focus groups that will be insufficient for drawing strong conclusions, as we were limited by the constraint of completing the research during the author's doctoral programme and the difficulty of recruiting multiple participants from one research group to attend a session in-person at the same time. Nonetheless, we present our findings as preliminary results, and as an example of a future direction to take the research presented in this thesis.

8.3 Preliminary Results

8.3.1 Evidence for the Existence of Communities of Practice

The team members in FG1 report that people work on aspects of the group's software because their group hierarchy mandates it, rather than out of passion; they discuss "compulsory PhD roles" in which the principal investigator tells students to take over areas of the software that are related to their research topics. While this may not seem like a fruitful place to look for evidence of a community of practice, the members report that working together on improving their software engineering skills helps them to cope with the software work, even if their principal investigator doesn't expect them to do so.

[FG1] 446 P15: . . . I know that [PI]'s not super keen on like software engineering

447 techniques for the sake of software engineering techniques

448 P16: yeah I guess but I think sometimes software engineering techniques must be

449 practiced for the sake of software engineering is the

450 P14: for my own sanity *laughs*

451 P16: yeah yeah basically

There are two RSEs in the group, and the participants reported that they mostly work on web tools and databases that the group describes as "products", to which commercial companies purchase access along with support from the RSEs.

The scripts, models, and other software that researchers create to support their publications is described as a "chaos" of individual Github repositories that may get migrated into a shared repository, metonymically named after the research group.

This dichotomy between what the participants see as well-supported products and chaotic research software parallels the pairing of order and chaos that Wenger describes as a basis for emergent structure in a community of practice. [190, pp.96–98] The observation of a difference between the software as supported by the RSEs and the software produced by individual researchers can be the impetus for perturbing their practice and adapting it.

We asked the group about the relation between these "products" and "chaos", and they drew a line on the whiteboard with the products at one end and the "chaos" at the other. They had a discussion which elucidated where on the continuum the RSEs got involved, and went on—led by participant 16—to discuss where hypothetically they would like to add RSE involvement.

The current, grass-roots approach to software engineering practice in the "chaos" region is evinced in the way that the group reviews their work through a barter system ("I'll get you a pint if you find a bug"). Two of the participants discuss a wish that there were more people "looking at the code, like, trying to use the code before it goes out", and discuss a barter approach to finding peers to review their work.

The participants describe their belief that the benefits of improving these practices come from increased use of the software among their research community, which leads to more citations of their work. In a group discussion reminiscing about how one of the group's PhD students created a well-regarded software package without being prompted ("nobody told him like you must now make this into an RD Kit package"), one of the participants tells an anecdote of two software packages released at the same time, from different research groups. Both authors worked on applying deep learning models to the same research goal, and published within a week of each other; only one of them released usable software. According to Participant 15, "That's why everyone goes oh so I use [software 1...] and no-one goes [software 2]". In making this connection, the participants join the software

community of practice they have formed in their group with the wider research community, and furthering the research goals of the group; publishing the software leads to greater impact for their research.

8.3.2 Participation in Communities of Practice through Software Use

An important difference between FG1 and the other focus groups is in how the participants consider their software mediates interactions with people outside the research group. For FG1, the software is available for people who don't have time to work on similar software themselves; either on a free website, or by paying "I think it's five grand a year" (Participant 15). As discussed in section 8.3.1, the group's RSEs mostly focus on the paid-for "product" software, and don't have capacity to help the researchers with the other software; so if the RSEs are contributing to research, it's research done by other organisations, not by researchers within their own group. This provides contrast with the discussion of well-engineered software leading to greater impact; the group is using its reputation to sell software written by RSEs that supports other organisations, rather than working with RSEs to improve the perception of their own research by creating better-quality software.

FG2 explicitly explore this dichotomy, discussing how re-using their software might conflict with another group's goal of increasing their own academic impact, or even profiting as the spinout company that exploits the software benefits from the collaboration.

[FG2] 356 P17: . . . few additions but back then it's like I don't know half or more than half of the

357 maintainers creating a company that kindof like making very clear that there is [software project]

358 there is [spinout] and they're two different things of course there's overlap but you know

359 but also think that we think how can we encourage other people it's in a way hard

360 enough to con encourage them to contribute a thing that like a researcher that me in my

361 researcher role I'll claim the credit and you know in increase my reputation on that even
362 worse is like oh I'll put my contribution in as a you know a contributor to the project and
363 it's going to be a company will be the er making money out of it's I think part of it was...

The participants in this group see this mindset as a difficulty to be overcome in increasing their collaboration with the users of the software, not a division or a reason to withdraw from collaboration. They include external developer collaborators in planning discussions, and promote some of them to a formal group of "maintainers" of the software, demonstrating the *legitimate peripheral participation* that is an aspect of a community of practice. [190, p. 100]

This legitimate peripheral participation is evident in FG3 too. Participant 21 describes community engagement as an explicit goal of the grant that's currently paying for the group's work, with one of their four work packages supporting community conferences and "workshop type things".

8.3.3 Requirements as Negotiation Within a Community of Practice

This variety in approaches to handling the research software's users is evident in the discussions they have regarding software requirements, which is one of the SEMAT alphas they encounter in the card-sorting activity. The description on the card for the Requirements alpha (see Appendix E) is generic, and each group's participants seemed to identify different sources of requirements as they tried to evaluate their importance. In FG1, the requirements that other people had to be able to use the software were described as negotiable, depending on whether the author wants the software to be re-used, while the scientific needs were considered mandatory. Participant 15 feels able to choose between making software usable by other people "or I wipe my hands of this", but producing the expected research goals is "not optional". This is consistent with the position above that the research group doesn't form a community of practice around its software with a wider base of users or contributors.

The group describes the software they produce as fitting into two categories: tools or scripts that group members use to enable their own computational research papers, which the researchers work on directly; or packaged "products" that commercial wet labs license to validate particular results, that the group's RSEs support. As such, their software appears to occupy levels 3 (domain-specific tools) and 4 (project-specific code) in the Hinsen stack [5], with the researchers mostly or exclusively working at level 4. Participant 15's comments about requirements are in the context of level 4 software—software that the researchers write to support a publication, and that they publish alongside the papers. As such the immediate user of the software is the researcher who's writing it, and the decision to make it "usable by other people" is a question of service to and impact in that author's research community rather than an explicit goal for the software project itself. Again, we see that the group's community is their research community, not a software-based community of practice.

Similarly, the members of FG3 seem to distinguish "the requirements of the project" from what other people might want the software to do, which they try to address by encouraging those people to change the software themselves. They view their software as a "service library"—equivalent to level 3 in the Hinsen stack—and if someone wants to put it to a particular use then "well tell us what you want, we'll tell you how to write it, but it's not like we're taking that requirements and developing the code" (Participant 20). Conversely, the requirements of the project are the tasks they agreed in the statements of work packages negotiated with their funders, "just updating the codebase" (Participant 19).

8.3.4 Stakeholders as Potential Members of a Community of Practice

In discussing the Stakeholders alpha in the card-sort activity, the focus groups demonstrated different relationships with their stakeholders that potentially include or exclude those people in a shared community of practice with the researchers. For FG2, encouraging stakeholders to become more engaged with the group is an outcome they seek to bring about, with the goal of improving the sustainability of

the software project. Participant 17 identifies that the success of the software was in "having spotted the opportunity of having an open source tool for [scientific goal] that is easy to use", and now sees the interaction between team and stakeholders as an important focus issue. This, along with the discussion of external maintainer roles and a steering council seen in section 8.3.2, suggests that the project is at least at level 5 (Pollination) in the Software Sustainability Maturity Model [8], and that the group's intentions are consistent with attaining higher sustainability levels. Elsewhere they discussed an ongoing plan to create a governance structure with a scientific advisory board, consistent with the highest levels (level 7, Ripening; or 8, Dispersal). In a follow-up discussion I asked them more about sustainability, and they also brought up that their stakeholders "are the ones who are gonna fund the project going forward" (Participant 18).

In the case of FG1, stakeholder management is handled by the PI who wasn't present in the focus group, and "speaks to so many people" (Participant 16). In contrast to the position taken by participant 15 in section 8.3.3, in which what other people want from the software is a factor they can choose whether or not to consider, at the outset of the card-sorting activity participant 16 identified stakeholder interest "and like thinking about if the people who are going to be using your code and how they might want to use it" as an immediately-relevant consideration which participant 15 agreed with. The specific considerations they mentioned were matching complexity to potential users' abilities, and how to release the software.

While this statement does suggest that the group considers how their software will be used, it doesn't indicate that the members see themselves and their software's users as participating in a shared community based on enthusiasm for the software. Similarly, the members of FG3 deal with stakeholders first in the card-sorting activity, but because they don't see them as important. This appears to be related to the group's position in their project lifecycle, where such considerations were critical earlier on but are now seen as "solved". As with FG1, they don't discuss any communitarian relationship with their stakeholders, and focus instead

on bureaucratic aspects of their relationship; holding stakeholder meetings with researchers whose PhD students were contributing to the project.

8.3.5 Reification of Abstractions in Research Software

In Wenger's theory of communities of practice, participation is accompanied by *reification*—the treatment of abstractions by members of the community as if they actually exist. [190, pp. 57–58] Through reification, people project themselves and their understandings into the world, and negotiate a shared meaning.

We searched for examples of reification in the focus groups by noting in the transcripts examples of participants talking about actions being done to, for, or by an abstract concept. For example, in this extract:

> [FG3] 388 P20: . . . then now you know I guess because because we
> 389 don't all have to publish or we don't all have to publish science in order to be rewarded
> 390 for software engineering then then you know these models have evolved and and so so
> 391 now um it's it's kindof OK to have the code that we're working on to be open to the
> 392 public. . .

Participant 20 talks about "science" as something that they can publish (but don't have to), and the implications for whether they publish their code to the public. Science here is an abstraction—a discipline of systematic organization of knowledge—so while their publication may be an example of a scientific approach to research, saying that they published "science" itself reifies the concept.

We choose to exclude examples of reification that coincide with the essence alpha attributes from [44]. As the participants had access to the cards from Appendix E even at the beginning of their sessions before we set the card-sorting activity, it is difficult to know whether reification of these concepts is part of their negotiated understanding of their situation, or is motivated by the concrete presentation of the concepts as defined in SEMAT that they encountered as part of the focus group.

In FG1, the participants divide members of the group into three sub-groups dependent on their specific focus within the group's research area. This division

is strong enough to stand in for the members' identities, as observed when two participants negotiate over who leads in the whiteboard activity; Participant 15 calls Participant 14 "you're sort of [sub-group A]".

The reification of research focus as identity is strong enough to survive one of the sub-groups no longer having anybody researching it. The participants start their task map by connecting their sub-groups to their principal investigator, and Participant 15 adds "[sub-group C] but no-one actually does that any more", still writing it in a box on the whiteboard.

Later, while completing the card-sorting activity, participants discuss the sub-group identities in relation to limiting collaboration in the group, while also questioning the need for collaboration. Participant 16 laments that nobody from sub-group A is present "to tell us what that is like"; the sub-group associations are so strong as to act as in-group/out-group labels. Participant 15 believes that the "open ended" nature of research software means "you should do it on your own" to avoid tripping over each other. Recalling that Participant 15 told the 'parable' related in section 8.3.1 of the two software packages where only the openly-available one now has wide adoption, we infer a "producer–consumer" separation that is consistent with this group not having a community of practice centred on software; the participant sees value in lots of people using their software, but not in contributing to its development.

Both FG2 and FG3 reify the concept of "technical debt". Ward Cunningham introduced the idea that sharing immature, first-time code is "like going into debt". [220] The programmer pays back this debt by rewriting code as they improve their understanding, but if they don't do so then the debt compounds. By reifying technical debt, these groups externalise the idea that a certain amount of their time must be spent on rewriting software that they already released, for example:

In the case of FG3, removing technical debt is part of a general effort "to modernise the codebase" (Participant 21) that is one of the work packages their project funding supports; it's also "eaten a huge amount of time, more than was expected".

Meanwhile, FG1 talk about the same work, but without reifying technical debt. Participant 14 mentions that "the maintenance is kindof precarious right, 'cos that just means being passed on to the next PhD student", to which Participant 15 replies that "this entire system used to be utter chaos because, like, it was written in a way that was kindof not very forward thinking[...An RSE] had to spend an ungodly amount of time rewriting the entire thing". In these interactions we see that the members of FG1 are further away from a software community of practice than those in the other groups, who demonstrate participation through their use of a shared argot.

FG3 also talked about "software engineering" and "scientific computing" as distinct backgrounds that a person could come from, with incompatible expectations. Software engineers are described as "people with, um, like, software engineering experience, but not much of a clue about scientific computing really" (Participant 20; Participant 19 nodded and said "yeah" repeatedly during this utterance). The decision to do test-driven development while working on their software was made by the software engineers.

The group reported a "friction" between these two backgrounds which I asked about, and this time the group connected the friction to the different choices of programming languages, and adherences to practices, coming from the different backgrounds. Those in "scientific computing" had experience of writing finite element solvers in Fortran, while "the software engineers" use Java, and those on the scientific side thought they would "lose out on a wealth, erm, something like 50 years of development of linear algebra code" by switching to Java (Participant 20).

Two of the participants went on to explain that the writer of the finite element solver came from a computational science background, and intended to create a Matlab-style front end to their software, while the two people named as software engineers both went into jobs in industry. This discussion shows that "software engineering" and "scientific computing" are considered distinct disciplines that lead to different ideas about how to do the work, and that different programming languages are considered appropriate in these fields. It additionally demonstrates

that in proposing new ways of working with research software and adoption of new tools, RSEs need to be sensitive to the existing experience and successes of computational research, as an attempt to throw the metaphorical baby out with the bathwater introduces tensions and division.

A similar dichotomy is observed in FG1, in which "research" and "engineering" are presented as conflicting domains, and while Participant 16 hypothesises working without the conflict, they also oppose the reifications "research" and "software", saying that they are "two things", which Participant 14 says "address different needs". In that group, at least, the division between a research approach and an engineering approach seems almost absolute.

8.4 Discussion

8.4.1 Summary of Findings

All three of the focus group sessions demonstrate aspects of communities of practice, including: participation as negotiation between members of the community; reification as construction of shared meaning; and fluid community membership with legitimate peripheral participation. The focus of the community of practice can be on software, as seen in FG2 and FG3, or on the research area, as observed in FG1.

Even FG1 demonstrates features of an internal community of practice around software, with participants describing ad hoc approaches to code review (swapping a pint for a bug) in the absence of formally-agreed mechanisms, and perceived antipathy from their principal investigator. The research group does include two RSEs, who focus on the "product" aspects of the group's software and don't get involved in the software researchers create. While participants did express a wish that their RSEs took more of a role in the rest of the software, they also reified "research" as a conflicting domain from both "software" and "engineering". Perhaps in groups that have little internal software engineering expertise or explicit software engineering practice, connection with a larger community of external software engineers would help them to develop a bigger collection of ad hoc practices and satisfy their preference for greater engineering rigour in their software—the

participants in this group did express that better-engineered software is more likely to get used and cited by other researchers.

Recalling the finding in section 8.3.3 that the software created by researchers mostly occupies layer 4 in Hinsen's scientific software stack, there would appear to be a potential chicken-and-egg problem in bootstrapping a software-focussed community. The software needs to be findable and usable by other researchers to gain a user and developer community—but unless the researcher-programmer is a "Proto-RSE" (see section 2.2.3) who considers those properties important and knows how to engineer them into their software, it would require input from the as-yet non-existent community to make the required changes in the software.

Meanwhile the other two focus groups work on software at layer 3, which necessitates making the software reusable and engaging with a wider community of users and potential developers; put another way, if the software were not reusable and a community were not formed, then the software would not find adoption as a layer-3 domain specific tool. Both of these groups deliberately set out to create reusable components that other researchers adopt: according to Participant 19, FG3's project was formed as an e-Science initiative, and the FG2 project started when "I first went to see [FG2 PI] and he said I'd like [FG3] for [PI's research domain] please".

As such, only FG1 evinces the evolution of software down the stack from supporting a particular project to becoming reused in a research domain. The discussion of two simultaneously-published packages in section 8.3.1 shows that the findability and documentation of the software were the inputs that support this transition, and increased publication impact was the reward. However, the group doesn't have a deliberate approach to enacting this transition, or to judging whether it's an appropriate direction for each of their publication-specific tools and scripts.

The turnover of staff on a research project leads to differing levels of focus on various aspects of the work, as in FG3 in which early collaborators on the project brought software engineering experience and practices that still define how the group works, even though those people left the group and academia altogether for jobs in industry. However in FG1, while participants recall a particular PhD

student who created well-written software without being told to do it that way, they also note a general lack of motivation to collaborate in their research group which suggests that relying on the "longitudinal community of practice" in which the group remembers the good practices of former members—the "institutional memory" or "organisation memory" discussed in section 3.6.3—is unreliable.

8.4.2 Answering RQ4

In section 4.2, we hypothesised (H4) that the "embedding" of RSE into a research group's structure would reduce the opportunity for the group to close the software skills gap by defining practices and outcomes for the software, as distinct from the group's research goals. This led us to ask RQ4, whether the structure of the relationship between researchers and RSEs in a group affects the research software practice.

Our focus groups evinced two different organisations:

- FG1 "embeds" two RSEs into the research group, which is based in a single institution and department.
- FG2 and FG3 each include RSEs seconded from dedicated RSE groups at multiple institutions.

We observed that the embedded RSEs in the group where FG1 participants work focus on the productised aspects of their software and don't get involved in the software written for research—a participant drew a line on the whiteboard demarcating the boundary beyond which RSEs "go like actually like we won't touch anything". The group brought up this distinction in a discussion of their principal investigator's position on software engineering: Participant 16 originally paraphrased the PI by saying "I know what [PI]'s opinion is on code which is that code is disposable", before correcting themselves to "code is cool but science is cooler was the exact quote that [PI] told me".

The participants see software engineering practice as important both for their own sanity and to increase the number of citations of their research, but application of those practices is inconsistent and something that they wish would see greater formalisation, regretting that they don't provide themselves an opportunity before publication to address maintainability and usability issues before publishing research software.

The organisation of FG2's work almost entirely contrasts with that of FG1. The current structure comprises maintainers who negotiate the day-to-day work using the Github software forge website, with a wider community of developers for whom the group's software is also the centre of their interaction. Alongside this maintainer system are a steering committee set up at the request of one of the funding bodies, and a spinout company that licenses the software to commercial clients. Where FG1 identified a desire to involve software engineering practice in their work, FG2 identified a need to get more direction on their scientific goals to ensure those are being pursued, and are setting up a scientific advisory board to provide advice. The participants see sustainability of their software as an important problem to solve, and discuss the problems they have bringing other research groups on board as they see broader participation in the software's development as the solution to that issue.

Meanwhile, the participants in FG3 are solely working on software outcomes, as their grant supports four work packages related to software development and building a community around the software. For this group, scientific goals have been externalised and are pursued by other researchers using the software, supported where necessary by the group we studied. The actual division of responsibilities can involve negotiation on the software development too, with Participant 20 indicating that if someone wants software changes to pursue a research question, the group will help them to understand how they make the required software changes themselves.

In this group, "the software developers run the project not the PIs, not even remotely . . . the software engineers run this project and we just ask questions every so often" (Participant 19). This delegation allows the RSEs to define both the software requirements and the way of working, and leaves them entirely free to

define the practices used in the team. The three participants described a set of "initial decisions about the language and everything else" — including use of test-first development and continuous integration — that are important for the team to "stick to", and that individual contributors are free to work however they want as long as they treat those rules as "principles" to adhere to.

We therefore see that the three different organisations of research group are associated with three very different approaches to research software:

- FG1: Two RSEs embedded in a research group—software engineering focus is limited to some specific software that is delivered as a "product" to certain stakeholders. Research software practice is ad hoc, transactional, and "chaotic".

- FG2: RSEs co-investigate the research goals—software engineering is led by a core team of maintainers who treat software sustainability as a critical goal. Skills in both software engineering and the research domain are seen as necessary, with the ideal contributor being someone with experience in both.

- FG3: Investigators lead grant applications to work on software goals but delegate group-running to RSEs—software engineering is the sole focus of the group, working within a set of agreed principles and with great autonomy over local practice. Research aims are externalised onto collaborators, who may be given help by the group to achieve their own goals but are not empowered to set the group's requirements.

What causal factors lead to these organisations are mostly not clear from the data, so we cannot say for example whether FG1 has two embedded RSEs because that is the staffing levels they need to achieve their software engineering goals, or whether the software engineering goals of the group are defined by the two embedded RSEs. What is particularly striking is that while the participants in FG1 talk about the benefits of better-engineered software in hypothetical terms and by comparison between existing examples, the other two groups identify

specific software-related improvements they need to make in their work, which both include community-building, and they discuss their plans to fund and execute those needs in concrete terms.

Both FG2 and FG3 show evidence that their members are embedded in a community of practice centred on the use and development of their software. Each group brought up its connections with RSEs and developers at other institutions, suggesting that these communities are spatially spread. For FG1, the community of practice is based on the scientific research that the group participates in, and the software mediates that practice (either via publishing papers that include their software, or selling access to commercial customers).

We investigate these findings in the context of the data we gathered in earlier chapters in 9.

8.4.3 Methodological Contributions

The theory of Communities of Practice provides an illuminating perspective on the shared negotiation of meaning undertaken by groups working on research software. We see how the participants work together on their practice, the areas of enthusiasm, and how the work follows or works around the job roles and reporting structures in the group, and how other people from outside the research group are involved. Observing the reified abstractions and legitimate peripheral participation gives us information about both the stability and flexibility of the communities of practice and their membership.

While others have applied Communities of Practice to software engineering education and described the challenges inherent in fostering communities of practice in academia, as we described in section 3.6.6, we believe that the work we present here represents the first application of the theory to the practice of constructing research software in academic settings.

The focus group approach complements the other techniques used in earlier chapters, by creating a one-to-few relationship between the researcher and participants distinct from the one-on-one context of interviews and the impersonal, web-mediated

interactions of surveys and blog-post analysis. By encouraging participants who work together to complete activities as a group, we see details of their interactions and relationships that might not come out in other contexts or that they may choose to present in different ways when their colleagues are absent.

8.4.4 Threats to Validity

The structure of the exercises that focus groups participated in led to certain topics being excluded from analysis. For example, the card-sorting exercise uses ideas that are reified in the SEMAT kernel, making it difficult to understand whether the participants also reify these ideas in their communities of practice, or have a different mental model (individual or shared) of software construction.

Additionally, the exercises focus on performance within the participants' research group, and as such do not naturally highlight broader participation or communities of practice beyond the immediate group. While we saw evidence of these communities in our data, it is possible that participants would have described more examples with further prompting.

The range of research group organisations in evidence in our focus groups does not represent an exhaustive exploration of the possibilities. We did not recruit any participants from post-92 institutions, or groups in which the software was written by external commercial contractors.

8.5 Further Work

This work can be extended by explicitly considering professional relationships that researchers make with people outside of their groups. One way to do this would be to form focus groups from people who work on shared activities across multiple institutions, for example members of professional associations or participants at conferences and workshops. Another is to broaden the range of research group organisations and host institutions from which participants are drawn, for example government research institutions and post-1992 universities. Different host sites will have different levels of on-site support for software engineering activities and as

a result researchers and RSEs at those sites may rely on external communities to different levels.

Having determined that communities of practice exist within research groups and between researchers and RSEs in different groups, including across different institutions, new work can explore the ways in which these communities negotiate and improve their practices. One possibility is to perform ethnographic fieldwork to observe how these groups function, and to compare those observations with the descriptions of group behaviour provided in the focus group activities.

As noted in section 8.2.1, the cohort included in this stage of the research is too small for our conclusions to be strongly defensible. We propose extending this work by running more instances of the focus group, to build more confidence in the results.

8.6 Conclusion

We facilitated focus groups with members of three different research groups who all create software in the pursuit of their research projects. In these sessions, participants collaborated on activities designed to elicit how they organise their work, and how they consider the importance of various aspects of software engineering and their opinions on their group's maturity at managing those aspects. We recorded and transcribed the sessions, and used the theory of communities of practice to analyse the participants' interactions, which we presented in this chapter as preliminary findings.

We found that all three of the focus groups we conducted demonstrated aspects of communities of practice that coalesced around their software development activities. In the case of FG1, this manifested as an internal community of *ad hoc* code reviews and tacit expectations for documentation and other code quality aspects, agreed among researchers with the goal of improving the citeability of their research by making their software easier to use. For the other groups, external use of the software was an explicit goal and they formed communities of software engineering practice with users, developers, and collaborators from external organisations.

We theorise that the distinction between these communities of practice is related to the structure of the research groups. Where FG1 has "embedded" RSEs tasked with working on particular aspects of the group's work, the rest of the group forms an *ad hoc* software engineering community to try to improve their software with no formal support. Meanwhile, the other groups use "pooled" RSEs who are based in a central RSE group, who therefore already have a professional network outside the immediate research group. In these groups, the communities of software engineering practice extend to people outside the research group with differing interests in the group's software.

With these observations in mind we now combine the results of each of our research chapters, to explore whether the conclusions on group organisation we reach in this chapter support the conclusions of research software engineering's liminal status and mimetic nature discovered in chapters 6 and 7.

9 Comparison of Findings Across Instruments

Contents

9.1 Introduction

In chapters 5–8, we described, carried out, and analysed a series of quantitative and qualitative investigations into research software engineering. Here, we combine these studies and analyse them together. We highlight areas where the topics analysed in one study arise in others, indicating whether the broader data agrees or disagrees with the conclusions drawn in the individual study chapter. In doing so we compare the analyses from each chapter, that took different theoretical

approaches as their bases.

9.2 Illuminating Survey Results with Qualitative Data

We take the quantitative results from our survey in chapter 5 and search for reinforcing or contradicting results from our qualitative instruments.

9.2.1 Distribution of Respondents

We noted in section 5.3 that most of the responses to the survey came from students, and from people working in scientific research disciplines. We examine the other data to discover indications of how RSE is distributed among the research community and to understand whether our survey suffers from a sampling bias or whether the population is skewed towards students in the sciences.

The Role of Students in the Qualitative Studies

None of the interview participants was a student. Participant 7 described a neuropsychology game that they designed with two other collaborators, but after publication "now we have a doctoral student who is more interested in ... moving this forward". They also described "one particular student" who came to them for guidance with writing software, who "is now actually a data scientist [at] a high flying company", and described software that they had worked on as a doctoral student.

Participant 12 described an HPC programme at a university they worked at in which one of the corporate sponsors "funded some Masters students and part of the funding requirement was that these Masters students would then do a bit of work, you know, kind of RSE style work."

In the blog data collected for chapter 7, students as a position in the academy are not mentioned at all. The word "student" only appears in the situational sense of someone who is receiving instruction, in the posts on in-person learning (Blog 15) and online learning (Blog 14).

Student participation in research software was prevalent in the focus groups. In one group (FG1), one of the participants was a D.Phil. student who wrote software in the research group. That group described doctoral students being assigned software components to "own" by the principal investigator, based on the relatedness of the software to their research interests. The "ownership" lasts until the student leaves, at which point the P.I. assigns the role to another student. Two of the participants agreed that the scope of this "ownership" role excludes usability concerns:

[FG1] 326 P14: . . . so these like

327 PhD students will be put in charge of like we'll go you are now in charge of this particular

328 database and not the containerisation not the making it pretty for the customers *turns

329 to P15* is that right?

330 P15: yeah yeah

Participant 17 explained that in the initial stages of FG2's research software, "a lot of the development, um like, it was a PhD student and then a postdoc", and that they have students applying for internships via Google Summer of Code which "really helps".

In FG3, two of the participants explain that the project was founded by a large collaboration with "10 PhD students and a couple of postdocs" (Participant 19) who had to start writing their research software because the package they intended to use to achieve their science goals was unavailable for licensing reasons. The researchers discovered "we had this massive grant which was worth about two mil two or three million quid and all these DPhil students and no software to work on cos the idea was we were going to make it an e-science platform and um so we had a choice we could either find another software package or we wrote our own . . . so we decided we were just going to go for it and teach people how to program" (Participant 20).

These examples show that students play a role in many research software projects, and support their prevalence among the survey responses. The absence of students in interviews and blogs is therefore surprising, given the broadly inclusive approaches to recruitment we took in each of those studies. We hypothesise that the

RSE community is focusing on supporting in-post practitioners[1], so that information shared in blogs and communities where RSEs discuss their work do not include students in the same proportion that they contribute to research software.

The small amount of data about students in the interview transcripts is likely to be due to the design of the interview, in which we focused on asking about the properties of professions (see section 6.2) which could lead to respondents focusing on RSE as employment.

RSE and the Arts, Humanities, and Social Sciences

The survey responses we reported in section 5.2 included only one response from each of the Humanities and Social Sciences divisions, with all other responses coming from Medical Sciences or Mathematical, Physical, and Life Sciences. This bias towards scientific disciplines is also evident in our qualitative investigations.

In the interview data, only participant 8 mentions research software projects in the humanities, and only to say that the people who review grant applications are "not super technically minded" so that project descriptions are "woollier" than in the sciences.

None of the blog posts we discovered mention software or research outside the sciences, and indeed Blog 12 states that "An RSE is typically a software developer with a strong background in science, engineering, or mathematics."

All three of the focus groups we engaged in chapter 8 work on projects in the physical and medical sciences.

We consider that having used very different sampling and recruitment strategies in each of the phases of this research, the consistent lack of evidence from arts, humanities, and social sciences is a systematic absence that reflects the underlying population. This finding reinforces our result from blog analysis in section 7.3.2 that RSE is primarily a scientific endeavour, and does not yet support other research disciplines. As discussed in that section, we consider that there is a need for research software skills in non-scientific disciplines, and that RSE is not yet meeting that need.

[1] Consistent with its mission "... to support the creation of an academic career path for Research Software Engineers" [83].

9.2.2 Mentoring and RSE

We found in section 5.2 that only two of 37 people reported mentoring by RSEs as a route to skills acquisition, the joint least-popular route among respondents along with formal training outside the University. While formal training (either in programming, or software engineering) within the University and self-learning were the most popular pathways, mentoring from a research supervisor or experienced researcher is much more common than mentoring from an RSE, with 12 respondents choosing it.

The interview and blog data that we collected mention mentoring among RSEs, including a formal mentorship program. Mentoring comes up repeatedly in interviews, but is only mentioned once in the blog entries. In describing "RSE postdocs" as part of their strategy to initiate an RSE programme at Stellenbosch University, the author of Blog 2 states that the "Society of RSE has graciously included one of this Professor's postdocs in their RSE mentoring scheme, thus helping establish the RSE identity here."

Interview participant 11 also mentioned this scheme, saying that "I think like software data carpentry is a great for teaching the tools. But I think we need mentorship as well with. Yeah, just like I mentorship network and that is again something that is coming down the pipeline from the Society of RSE as well."

Participant 6 describes pairing inexperienced "some of the more junior RSEs with more experienced ones", both to transfer skills to the junior, and because "it's quite good for" the senior RSE's career development.

Participant 3 also described a mentorship relationship between senior and junior RSEs, saying that the difference between two pay grades on their RSE career scale is "are not really technical differences right? They're the more on the sorts of responsibility mentoring [...] management side of things."

Finally, participant 8 describes looking for "books or kind of blogs and stuff [...] about thing like coaching people or mentoring people", as they had seen those skills in a senior colleague that they were shadowing. This shows how mentoring can be part of a blended learning experience; the participant observed that a senior

colleague mentors others, and turns to published literature to learn more despite the existing relationship with the mentor.

In the interview context, mentoring appears to relate to experienced or established RSEs helping juniors to network and acquire skills, not to RSEs supporting researchers with software-related problems.

Meanwhile, in the focus groups we discovered limited evidence of RSEs mentoring their colleagues. One group (FG3) uses mentoring as a way to reduce their own workload, training other researchers to integrate their code into the group's project so that the group doesn't adopt the work itself. In another group (FG1), the participants indicate that the group's RSEs focus on particular "product" aspects of the group's software, and the researchers have an *ad hoc* approach to software engineering practice that they negotiate amongst themselves.

In our survey instrument we did not provide a definition of "mentoring", and it is possible that respondents do interact with RSEs in ways that would fit some definitions of mentorship but that they didn't recognise when completing the survey. Carmel and Paul distinguish formal mentoring programs from "self-selected mentoring", in which a mentee seeks out a mentor and informally engages them to negotiate a relationship [221]. Holt and Lopez cite works by Nora and Crisp in defining four activities performed by college mentors:

- psychological and emotional support;
- degree and career support;
- academic subject knowledge support; and
- provision of a role model [222].

It is possible that a researcher/RSE relationship only partially covers these activities and that this restricted interaction was not perceived as mentoring by survey respondents. Another explanation is that there truly is not much mentor/mentee interaction between researchers and RSEs. This may be because RSEs are not engaging with researchers in a mentorship relationship, or because

the number of RSEs is too small for many researchers to get exposed to them as mentors. These two factors could combine, in a situation where researchers and RSEs collaborate but the RSEs have too much software work to spend time mentoring their colleagues.

9.3 Comparison of Blog Analysis with Interview Coding

The finding from section 7.3.1, that RSE has a lot of structures and institutions in common with academia, is consistent with the analysis of the interview data in chapter 6, in which we found that RSE is a liminal profession that seeks to distinguish itself from a "traditional" research career, but is not wholly separate from that profession. In the interviews, this was seen to lead to challenges of "service or subservience", as RSE tried to define itself and escape liminality as an "academic support" role. In the blogs, we see that RSE organisation is strongly isomorphic with the organisation in research, depending on the same institutions for funding and legitimacy. Additionally, this legitimacy is predicted by DiMaggio & Powell's hypothesis B-2 [188], that the reliance of a field on state support would lead to isomorphism: RSE depends on the same government-funded research landscape as the rest of academia.

In section 7.3.2 we found that blogs about RSE discuss only a limited subset of knowledge areas from the software engineering field. Compare this with our finding from interviews in chapter 6, that RSEs were generally expected to understand "the core software engineering body of knowledge". We saw in that chapter that learning on the job is an important route in the field. Participants reported learning "through osmosis", and using "a lot of self-taught skills". Given the background of self-taught computing skills and ad hoc knowledge-passing among research groups described in chapter 2, and confirmed in interviews in chapter 6, we theorise that adopting the logic of software engineering is a mimetic process among researchers. Knowing that they have problems managing software-related aspects of their work, and that the practices propagated within their research domains have been insufficient

to mitigate these problems, researchers look to examples of successful software organisations and copy the observable practices.

The missing attributes of the SWEBOK are consistent with the results in chapter 6, in which we discovered that RSEs report problems with managing maintenance work, and software requirements, two of the software engineering KAs that were absent in the blogs. In discussing this result with my supervisors, Prof. Gavaghan noted that in many research groups, the software developer and the "customer" is the same person, so maybe they perceive no need to manage requirements without no communication between different actors. We consider that while that may be viable in the short term, it affects the longitudinal sustainability of the research software because the effort will lack a written record of what the software was designed to do, and why it was designed to do it. Researchers could consider any potential reader of a paper in which the software is published or used as a stakeholder in the software's lifecycle. From the perspective of responsible research and innovation (see section 2.4.5), these stakeholders deserve input into the software development and maintenance.

We also see an important difference between the two categories of data. In interviews, participants talked apparently freely about the challenges of working with academics, for example reporting "falling out with academics" who "treat [their postdocs] horribly", and discussing the "upstairs/downstairs" class division in academia. On the other hand, the blogs are typically upbeat, describing RSE as providing "a clear sense of direction" for one author's career within academia, and "a critical part of the academic ecosystem" (Blog 2).

We suggest that this change in tone is due to the performative nature of blogging. Authors are claiming legitimacy for particular logics and trying to build their own reputations or those of the institutions they represent and support, which is better done by talking up the benefits and minimising the drawbacks. In one-on-one interviews, particularly with a researcher and RSE who they can recognise as "one of their own"[2], individuals can be more unguarded about sharing their experiences.

[2]As mentioned in 6.4.3, the author has worked as a senior RSE and may be perceived by participant RSEs as "one of us".

9.4 Institutional Legitimacy in Interview and Focus Group Data

In chapter 7 we used neo-institutional theory to analyse public blog posts about RSE, looking for evidence of institutions proposing competing logics for how RSEs work, and of people reconciling conflicting logics by evaluating the legitimacy of the proposing institutions. We found that the institutions of academia are strongly present in RSE, as RSEs legitimise their presence in the same employment contexts and supported by the same funding sources as "regular" researchers. Additionally, the logics of software engineering are inconsistently observed, with the knowledge and skills associated with software engineering only being partially in evidence. We turn to our other data to corroborate or refute this view of RSE as primarily aligned with the institutions of research, with software engineering second.

9.4.1 Software Engineering Knowledge Areas

We search for evidence of some of the areas of software engineering knowledge that were absent in the blog data, to see if our other sources provide insight on how RSE treats these aspects of engineering.

Software Requirements

Requirements management—both identifying initial requirements at project inception and adapting to changing requirements as a project continues—was absent in the blog data, but is a frequently-reported source of problems by interview participants.

In response to my question "What do you think the problems that RSEs commonly face in their projects are?", participant 1 replied:

> [Participant 1] So the main one is no requirements. So academic software, generally evolves, fast and. Sometimes academics don't have an understanding of you know how easy or hard it is to do particular software development. Uh. But yes, requirements is definitely a thing. It's it's the nature of the work itself that you know. It's hard to get the requirements, and since

> we don't have a profit motive we it's also hard to measure when you're succeeded or like, or how much you've succeeded in some sense.

Participant 2 reports "an awful lot of times" when they create prototypes to demonstrate to researchers who then say "I don't think I explained myself in in terms of what I actually need it to do, um because you've ended up making an assumption that sort of isn't true, or they say that's exactly what I asked for, but it's not now what I want or something like that." This shows that while communicating requirements is a problem, the act of getting early feedback via a prototype mitigates some of the difficulty.

Participants 3, 5, 6, 7, and 8 all also described difficulties with agreeing requirements with researchers.

Requirements were an explicit topic of discussion in the focus groups, because Requirements is one of the SEMAT essence alphas that participants encountered in the card-sorting activities.

When FG1 encountered the Requirements alpha, participant 14 thought it was "quite important" and "would put this up top" (of a rank of the alphas by importance) but participants 15 and 16 overruled this, saying that stakeholders are more important. Participant 16 said "I feel like also like the requirements like they might have like some basis as a result of like the stakeholders but they can change at a moment's notice", which seems to be a hedging statement (the use of "I feel" and repeated use of "like" suggest that the participant is not confident to assert this position as a fact), but this argument was carried and participant 14 concluded "OK so lower down then" to which the others agreed.

The members of FG2 indicated that requirements were not an important consideration for them, because the participants felt that identifying the requirements had been solved at project inception: "we've achieved the requirements really" (Participant 18). Participant 17 goes on to suggest that there might be future requirements, but that they "are more on the either we've spotted an opportunity or I guess fairly minor" and that they consider the requirements to be "already met"; the participants place future requirements "down the bottom" of their priorities list.

The final group, FG3, struggled to prioritise requirements, as participant 21 says that "obviously they're important but they could almost go everywhere" (in the ranking of alphas). Participant 19 echos FG2 in saying that "they're almost hard wired in that we know what we need to do", and in expanding on this says "I don't quite know where to put them because they're fundamentals of the project but kindof set and at the same time *laughing* malleable in the sense that if we find something else different then we do what we need to do so it's a bit tricky that one".

This broad range of opinions on software requirements—generally problematic in individual interviews, with groups reporting that they are unimportant, solved at the project inception, or expressing ambivalence—suggests a lack of systematic approach to requirements engineering among RSEs. The absence of public discussion of requirements in the blog data supports this conclusion, because for an author to publish their thoughts on software requirements to an RSE audience would imply a confidence on the subject sufficient to share their opinions, and an expectation that the audience is receptive to the topic. Alternatively, the blog authors could all consider requirements to be a solved problem and uninteresting to talk about in a public post—given the difficulties RSEs described with managing requirements in our interviews (chapter 6) and the "pain" of requirements management in scientific computing reported by [126], this interpretation would require that blog authors writing about RSE are a different population from RSEs, with different knowledge and experience.

We suggest that the practice of rapid iteration with fast feedback discussed above is most suitable for software at layer 4 of the Hinsen stack, where the immediate user community is in the group that writes the software. In the most extreme (but common) case, one person both develops the software and uses it in their own research, so communication to understand requirements is near-zero; not exactly zero, as peer reviewers and researchers who read papers based on the software may want to run or inspect the software to validate and replicate its behaviour.

At lower layers in the stack, the lack of a well-understood requirements management methodology normalised in the RSE community is cause for concern. Such

software has a larger user community with disparate needs, and can be critical to those users' research. Without a robust approach to managing requirements, RSEs risk introducing regressions into researchers' workflows, expending effort and resources developing features that their user communities don't need, and perpetuating the fragmentary nature of research software development as researchers write their own tools at layers 2 or 3 due to a lack of confidence in existing solutions.

We propose that RSE practitioners can learn from requirements engineering (RE) in both software engineering practice and research, to develop RE maturity in research software.

Software Maintenance

Like software requirements, software maintenance is a software engineering knowledge area that did not receive detailed coverage in the RSE blog posts we analysed in chapter 7. Two posts, Blog 11 and Blog 16, talk about maintenance of open source projects and packages, but do not connect the area with RSE. The only reference to RSEs engaging in software maintenance is a mention in Blog 12: "[Research Software Engineering] incorporates a broad range of activities such as software design, development, testing, maintenance, and optimization."

In the interviews, participant 3 said that a problem with maintenance was resource allocation: "I think in that sense of our ongoing support and things like that, and I think researchers often assume that they can come back and ask you more questions in the future, and you're like. I [am] working on [a] different project now no time you know? I can be helpful to a certain level but I have no set set aside time for this."

Participant 6 agrees, and says that in their group in a government research institution, the problem of paying for software maintenance is specific to research software engineering. Traditionally, their department was "responsible for software and maintained it and supported [it],", but the arrangement for RSE is that "you're paying for the person's time during this project," and if their (internal) clients want ongoing support, then they have to provide more funding. In this way, their

arrangement seems similar to that of RSEs in academic institutions, supported by particular post-doctoral research contracts to work to specific project goals.

Participant 8 reports that their group has experimented with costing website hosting and a small amount of staff time—"that's for not for maintenance, but for emergency"—into grant applications. They say that "actually there was no kind of kick back from the researchers, because when you say, do you want that release that website to be available after the project ends? They say yeah", but that at the time of the interview no project that they had costed on that basis had been awarded a grant so that the idea was still untested. Also, note that they explicitly excluded software maintenance from this model for ongoing costs.

Participant 1 deliberately connects the institution of academia with a logic of not considering software maintenance. They say that "in academia, I don't think maintenance is important in the general case because most papers are meant to be one offs." The FG1 focus group show a similar thought process, once software is associated with a publication it's "lost to the tales of time" (Participant 15) and even other members of the group might need to put months of work into getting the software to run again, if they try to use it later.

This points to a similar chicken-and-egg problem as that discussed in section 8.4.1. In that section, we considered that it would be hard to bootstrap a software community, because researchers publish layer 4 software on their own to support their papers, so the people who could contribute to and improve the software might not be able to find it or use it.

Here, the problem is in bootstrapping the expectation that software be reusable and maintainable. The organisations that support RSEs seek to improve software engineering skills application in research (see section 2.4.1), but the logic among practitioners is still rooted in "publish or perish", with the publication of a paper representing the end of any work on its materials.

The other focus groups, which work on projects with software creation as an explicit goal, do show evidence of engaging with software maintenance. FG3 is a purely software-based project, funded to develop the software and a community

of users rather than to achieve scientific goals, which the group externalises to collaborating or client researchers. As such, software maintenance is an important part of their work. The three participants, led by participant 21, described the four work packages that their grant funds:

- Migrating infrastructure from on-premise hardware in the university that the project started in, to cloud-based services that collaborators from other institutions can access.
- Modernising the code and updating libraries and tools that the project depends on.
- Adding new functionality and extending existing features.
- Hosting events for the software's user community.

Three of these work packages address needs identified in [216, Chap. 5 §1.3] as reasons to perform software maintenance: improving the design; migrating legacy software; and implementing enhancements. Additionally, Participant 21 gives a specific example of how a decision on how to design their automated tests led to higher maintenance costs, as the tests depended on the behaviour of the random number generator in the Boost library (a general-purpose utility library at layer 2 in the Hinsen stack), so any time the library changes its implementation, the project team must investigate and react to test failures.

Finally, FG2's people structure themselves around software maintenance activities, with "maintainers" being the key role comprising the core of the team who prioritise and largely implement the work on a day-to-day basis. While there are other parts to the organisation, including a steering council that takes strategic decisions and organises community events (see section 8.3.4), "many of these people *points to "steering council"* are maintainers but we'll let the maintainers decide it's more like the rubber stamp *fist to palm gesture*" (Participant 17).

As with FG3, the participants discussed how the maintenance burden of the project is dependent on the history of its development. Participant 18 connects this

with the project's maturity level, saying "all the maintenance stuff really starts to dominate... and support and stuff like that starts to dominate, and I think what suffers is the longer term direction and feature set" as the growing user community increase support demands, and report more bugs in the software.

As with software requirements, we find a wide range in approaches to dealing with software maintenance among RSEs, from maintenance being an explicit goal of a project to being explicitly rejected as an aim by participants who do not see that software has any need to update beyond publication of the paper in which it appears. We also see that even where software maintenance is perceived as important it is work that must be staffed and funded, and that receiving this funding is not always successful.

In section 7.4.2, we discussed the challenge that producing reusable research software opposes entrenched logics about recognition in academia: the software creators risk losing "credit" to researchers who publish results based on the work that went into the research software; the researchers risk having to share "credit" with the software creators, which they would not do if they wrote their own tools rather than selecting existing software. Nonetheless, both FG3 and FG2 initiated software-focused projects within the academic ecosystem and attracted research funding; and FG1 have software that is originally created for specific research papers but that gets "promoted" to the status of reusable projects.

All of the groups find software maintenance a challenge within this system, with FG1 expressing uncertainty over whether software should be modified at all after publication in a paper, and both FG3 and FG2 indicating that the maintenance burden grows to take an appreciable fraction of the project resources as the user community grows and technical debt accumulates.

There is therefore scope for RSE to adopt more efficient maintenance practices, including preventive maintenance that avoids the accretion of technical debt; and to identify ways to determine whether software can be archived after initial publication, or needs ongoing investment. This latter decision is particularly important in the case of software that creators conceive to support one particular experiment or research

project, and that goes on to find broader adoption within the relevant research community. Using the Honeyman model of research software intent presented by Goble [223], this decision supports a "transition" from one-off or prototype code towards a professionalised, reusable research software product. The transition is made more challenging because it may not be a deliberate decision on the part of the software author; it could be that other researchers discover software that the author intended to be one-off, and start reusing it and integrating it into their workflows.

9.5 Communities of Practice in Interviews and Blogs

In chapter 8, we found that people working on research software display evidence of participating in communities of practice, and that those communities can focus on software engineering practice or on the research domain. The evidence included participation as negotiation; reification as construction of shared meaning; and fluid community membership with legitimate peripheral participation. We look to our other sources of data for evidence that supports or contradicts these findings.

In their interview, participant 1 considers the community around research software as a distinct outcome that the creators must choose whether to engage with as they prepare their research for publication, which has implications on the software engineering practice. They find it "very strange in terms of collaboration" for an open source project if the software is published in sync with associated papers, with no other shared progress, "not that it can't work". They observe that for such a project, "many of the things that research software engineers are good in wouldn't apply there, like version control", indicating that RSEs can be selective about where they apply their skills to maximise impact.

We suggest that this community—and therefore, its potential impact on how the software's creators work—may develop organically without the creators' intervention. FG1 described this scenario in 8.3.1 when they said that two groups published software with similar capabilities around the same time, and researchers gravitated towards one of the two packages because it was easier for them to use. Thus, while

the creator may make the decision over whether their software is intended for the community, they should have the tools to consider this decision systematically and access to appropriate funding for the work generated by supporting the software—as multiple participants reported in section 9.4.1, it can be hard to acquire resources for maintaining software.

Participant 3 makes mention of three distinct communities in their interview: the "HPC community", which their department was not seen to be a part of before participant 3 made explicit attempts at integration; the "[software] carpentries community" which the participant sees as a good model for "getting people together and having a chat about things or where we've got a common interest and pushing that", and the "RSE community" which evinces "this huge, explosive growth where it's really enthusiastic and all that sort of stuff [...] but my concern is how do we translate that into actual changes and careers, right?"

Many of the other participants also discussed a community around research software, whether they name it as the "RSE community" or something else. For example, participant 5 has a role within the SSI, and describes their community team which is "funded by all 7 research councils" and "builds a fellowship and tries to spread the word about good software practices into the research domains". This funded work is centralised community-building organised by the SSI but being based on "fellowship" and "good software practices" it does represent a community of practice.

Participant 6 describes the experience of engaging with this SSI-led community, saying that they got involved through becoming "a Software Sustainability Institute fellow" in about 2018, meeting other like-minded people through their events. The participant continues to discuss getting involved "at the beginning of their [i.e., the SSI's] RSE leaders network, which is a sort of mutual self-help group for people trying to set up and run RSE groups." This network represents a more focussed community of practice than a broad "RSE community", being specifically for people who lead groups of RSEs. This experience demonstrates legitimate peripheral participation,

as the participant engaged with the community when they "first kind of got in touch with it all", at the beginning of their time as team leader.

Participant 6 also described how the RSE group in their institution acts as the focal point for an internal research software community of "people working on software in whatever their different roles are. This internal community allows RSEs to engage interested researchers and other peripheral participants in sharing software engineering practice. However, we refer back to the analysis of the survey in chapter 5 which indicated that most people do not acquire software engineering skills from mentorship by RSEs, and leave the question open of how much these internal communities achieve in terms of normalising and advancing software engineering practice.

Both participants 6 and 7 were involved with the RSE community from the beginning, as participant 6 says it was the "earliest days of a national association" when they started attending the SSI unconference, which led to them co-chairing the committee of the national RSE association. This committee "had been a fairly informal affair", which participant 6 helped to turn "into something that was a legal entity because it started to need to be able to do things like handle funds for the conference." Similarly, participant 7 "went to the first Manchester conference... and then I decided I could run to be part of the committee... when it was moved from being a collection of friends to something more formal."

In this shared experience, both of these participants took a role in reifying the "research software engineer" abstraction, turning it from an informal association of people with shared interests into a defined society.

The blog data includes multiple examples of community-building. For example, Blog 7 lists a conference talk entitled "Scientific Software and the People Who Make It Happen: Building Communities of Practice". The author of Blog 2 describes giving a talk at the Society of RSE conference that connected them "in person with members of the UK RSE community" which increased their motivation and gave them fresh ideas. Finally, Blog 12 explains that the site's section on RSE "delves into the RSE career paths and the RSE community."

Across all of these qualitative instruments we see evidence that communities of practice are an important factor in how the RSE role perpetuates and in how RSEs convey information between each other and their collaborators.

9.6 Conclusion

Comparing the data we acquired in the different phases of this research leads to a nuanced view of RSE as a profession. We find that students are important contributors to research software, but are unrepresented in discussions of the community and career paths. We also find a consistent lack of discussion of the arts, humanities, or social sciences in relation to RSE.

We identified the ambiguous position that while communitarian knowledge-sharing is important to RSEs, including communities of practice that bring RSEs together with researchers, people who make research software mostly do not consider mentoring from RSEs as a route to skills acquisition.

We consistently observed that the structures and expectations of academia shape the work of research software engineering, and that software engineering itself is of secondary importance with some knowledge areas inconsistently addressed. We conclude that software engineering is not actually systematically applied within RSE, and that personal preference or experience dictates how fields including software requirements and maintenance are approached and even whether research software is worked on at all, outside the "publish or perish" paradigm of scholarly papers.

In the next chapter, we situate these findings within the UK academic landscape and make recommendations for research software engineers and for further study.

10 Conclusions and Further Work

Contents

10.1 Introduction

In the preceding chapters we have shown via a mixed methods study that RSE predominantly supports people working in scientific research, even though other

academic disciplines rely on research software and significant contributions to research software are made by students. We have shown that RSEs mentor each other and support career growth within their field, but did not find evidence of RSEs mentoring researchers and supporting their acquisition of software engineering skills.

We reported that RSE is in a "service or subservience" relationship with academia, as it depends on the same structures and sources of funding and inhabits the same political spaces, in the shadow of traditional researcher careers.

We also discovered that the field does not apply software engineering knowledge consistently or completely to its work. We found that RSEs and researchers form communities of practice in which they share enthusiasm for and knowledge of their work, and that these communities of practice may have research software as their basis or may focus on the research domain.

In this final chapter, we bring together the results from the preceding chapters to answer our research questions. We present practical recommendations for the future development of RSE that are based on our evidence and situated in the context of the background discussed in chapter 2, and can serve as a "manifesto" for RSE in the same way that the Manifesto for Agile Software Development addressed the context of the software crisis.

We summarise the various contributions that we have made in this thesis. We indicate how future research can build on the work we presented in this thesis, and provide concluding thoughts.

10.2 Answers to Research Questions

In chapter 4 we asked four research questions, based on hypotheses we drew from our study of the background of RSE and the available literature in chapters 2 and 3. We now collect the evidence we gathered in the four phases of our mixed methods research study to answer those questions.

10.2.1 RQ1

We asked whether UK-based researchers have the skills and training they need to fulfil the software-related requirements of their work. In section 5.3, we argued that the results of our survey reveal that this knowledge and training are lacking. Turning to our qualitative research, in section 9.2.2 we found that "mentoring" in the context of RSE typically means senior RSEs supporting more junior colleagues, and that we found no evidence of RSEs mentoring researchers. Further, in section 9.4.1, we discovered that even among RSEs, some areas of software engineering knowledge are inconsistently applied, or even rejected by practitioners.

We conclude that despite the education programmes and community hubs described in section 2.4.3, and the evidence of communities of practice among researchers and RSEs explored in section 8.4.1, the "software chasm" posited by Diane Kelly in [107] is real. Scientific computing (remembering our finding that RSE mainly supports the sciences) and (industry) software engineering do have distinct practices and knowledge.

10.2.2 RQ2

We asked if RSE exhibits the following four properties, which are attributes of professions in the functionalist theory: a body of expert knowledge; technical autonomy; valuing service to others; and privileged status.

In section 6.4.1, we noted that the properties are only partially in evidence in our interview data, with the lack of a clear body of knowledge for RSE being consistent with our finding of the existence of the "chasm" between scientific computing and software engineering. Further, RSEs described their software skills as self-taught, or learned "through osmosis". In section 9.3 we show that this finding is consistent with the neo-institutional view of RSE we gained through studying blog posts, and suggested that software engineering knowledge is mimetically imported into RSE as practitioners see needs rather than being systematically adopted through training or education.

The other professional attributes were seen in section 6.4.1 caught in the power struggle of "service or subservience" with traditional academic structures and roles. In section 9.3 we observe that the organisation of RSE is isomorphic with research, and that RSE has not achieved autonomy or status relative to the academy.

The question of service to others is complex and caught up in the "service or subservience" dichotomy—RSE is perceived by its practitioners to be a role that supports research, but questions of recognition and status challenge this perception. Professional ethical responsibility appears in the literature, for example in [88, §3], but as noted in section 6.3.3 questions of professional service are generally thought of in terms of service to research and the struggle for recognition for software contributions to research, not service to a wider society.

We conclude that RSE does not have the status of a profession within the functionalist model, and is instead a role within the professional academic structure that requires some specialist technical knowledge, that has yet to be conclusively defined or propagated amongst its practitioners.

10.2.3 RQ3

We asked how the institutions and logics of research influence the way an RSE performs their work, and what competing institutions vie for legitimacy.

In section 7.4.2 we found that RSE can be perceived as a small collection of change agents trying to bring about a culture shift in academia, but who must adopt its logics and appeal to its institutions to remain legitimate and comprehensible. This need to thrive within the academic ecosystem makes RSE isomorphic with academia as it must support the normative values of the academic culture and career system to appeal to people who are embedded within that system.

This perspective reinforces the conclusions presented in section 10.2.2, that RSE is a liminal occupation in the shadows of traditional academia.

10.2.4 RQ4

We asked whether the structure of the relationship between researchers and RSEs in a research group affect the research software practice.

In section 8.4.2 we saw preliminary evidence of a relationship between the organisation of a research group and its software practice, though were not able to draw conclusions about the causality of the relationship. Using the theory of communities of practice, we observed that the researchers in a research group that directly incorporates RSEs formed a community of practice around the research publications. Meanwhile, those groups that produced research software consumed by other researchers formed communities of practice around the software and its development.

While this is a tantalising preliminary finding, we did not investigate a wide enough sample from the population of research groups who work with software to draw a firm conclusion, so we consider this question unanswered.

10.3 Recommendations

Combining the conclusions summarised in section 10.2 with the context described in chapter 2, we make the following recommendations for organisations and individuals involved in RSE.

10.3.1 Funders of Research Software Projects

We observed in section 2.4.1 that software sustainability commands approximately 0.02% of the UKRI budget, however in section 5.2 we found that 100% of researchers who responded to our discovery considered research software at least somewhat important to their research, meaning that all research outputs are potentially at risk due to the "chasm" between research software and software engineering practice we identified in section 10.2.1.

A first step in reducing this risk is to identify which research truly needs to be "sustainable", and which needs to be simply "available" for other researchers to

replicate the results from associated publications. Psomopoulos points out that while software is ubiquitous in data-intensive scientific domains, all software is not of equal importance [224], so RSEs need to identify where to spend their resources and effort. As we saw in section 9.4.1, this decision is being made locally by researchers and RSEs on many projects, based on their interests and values, with widely different outcomes. A checklist approach that researchers can readily follow, and that reviewers of funding proposals can quickly check, would allow research funding councils to readily assess the software sustainability needs of any project against a systematic standard and ensure that funding is spent on research software in situations where it would have the most impact.

The checklist should help stakeholders to assess the likelihood that other members of the research community—and, recalling RRI from section 2.4.5, other people impacted by the research—would use the software. Example questions could include whether the software is a one-off script written to support the current project or has more general applicability, whether it was created from scratch for the project or adapts software that's already available to the community, and whether there already exist other software packages that address related needs in the research domain.

For projects where the checklist indicates that the software is likely to need sustaining, all stakeholders in the project need to accept that the ongoing maintenance of the software will require resources and plan for those resources to be available. This could involve costing an agreed period of follow-on sustaining work into the project under consideration, or transfer of software development to another group or organisation (again, with the financial resources required to support its maintenance).

10.3.2 Software Engineering Skills and Education

With resources in place to address the "chasm", RSE leaders can attempt to improve the application of software engineering to research by systematically understanding the gaps in software engineering knowledge and skills application and introducing appropriate training interventions. We suggest that this is best approached by the

research councils each surveying the skills needs and capabilities in their own fields, to be sensitive to our findings that: RSE currently predominates in the sciences; and that researchers form communities of practice within their own research fields, so software engineering knowledge may currently be transferred through these communities as distinct "cottage industries". An analytic approach to the skills gap would move RSE away from a mimetic borrowing of visible practices in industry software engineering, turning it into a discipline that applies engineering knowledge as needed to the problems that researchers face.

As seen in section 9.4.1, requirements management is a software engineering skill that RSEs particularly struggle with. Improving requirements-management capabilities would increase the chances that reusable research software products can succeed in their research communities, driving the development of reproducible, higher-quality research software and results that are the stated aims of organisations such as the SSI. In industry, software requirements management (and the associated field of business analysis) is a mature domain, with practices that run from understanding the problem domain and identifying objectives to effectively managing changes to requirements [225].

We suggest that practitioners start by adopting, gaining proficiency with, and normalising standard industry practices, before teaching requirements management to their research communities and turning to the research literature on requirements engineering to discover innovative approaches that could support the research software context. A common objection is that in research, software requirements are unknown upfront because the work is exploratory [109]. Industry software engineering has developed iterative, incremental methodologies under the banner of "Agile software development" (see section 2.3.3), which allow for experimentation, feedback, and emergent development of requirements.

Felderer et al. identify a need for a new domain of "RSE Research" that investigates and shares better ways of creating software in research; they suggest requirements engineering as a target topic for the nascent research domain [226]. Heroux similarly suggests that requirements analysis and elicitation is an important

part of "Research Software Science", along with user-centred design [227]. Corral-García et al. propose a software development methodology for scientific applications that includes tracing and monitoring to identify changes in the requirements and notify developers [228].

Similarly, in section 9.4.1 we showed that RSEs have a wide range of experience and attitudes toward software maintenance. The decision of whether a given research software output even needs maintaining is related to the question of whether that software needs to be sustainable, discussed above. For software that should be developed in a sustainable fashion, a mature approach to requirements management helps RSEs to understand what changes are desirable, feasible, and what the resource needs are. Given the likelihood that existing research software was developed with an incomplete understanding of software practices including design patterns, testing, and documentation, developing the abilities to understand and make changes to "legacy code" [229] without introducing regressions in its behaviour are important skills for RSEs to adopt and propagate.

Jiménez et al. show that open source software practices improve the community-building and collaboration that support ongoing reuse and maintenance of research software, in addition to aligning well with the goals of FAIR software [230]. We suggest that RSEs should adopt an "open source by default" approach to producing sustainable research software, adopting the Software Sustainability Maturity Model [8] as a guide to their maturity level.

As we discovered that RSEs discuss mentorship among their own occupation but that other researchers do not learn software engineering through mentorship by RSEs, we suggest that RSE-centric organisations including SSI and the Society of RSE could formalise "outreach" programmes in which RSEs mentor researchers in software engineering skills. The goal of such a programme should not be to encourage researchers to consider RSE as a career path (though that might happen), but instead to increase the base understanding of software engineering knowledge among researchers. Such a programme would start a virtuous cycle effect, in which RSEs would be free to innovate in other areas of software engineering that

researchers have not yet broadly adopted, and then to transfer knowledge in those areas via mentorship, and so on.

Such mentoring programmes may already be in place on an informal basis, so the societies named above can maintain contact with the regional interest groups (for example, RS London and RSE Midlands in the UK) and peer national associations (Cohen et al. identify RSE communities in Germany, the USA, Australia and New Zealand, France, and the Nordic countries [231]) to discover and propagate emerging practice.

10.3.3 Professional Organisation

Research-producing organisations can reduce the need for RSEs to "play academic" and fit into the expectations for traditional research roles by providing distinct job descriptions and career structures for RSE roles. These structures should map to the pay grades and promotion pathways that the institutions use for academic careers, to mitigate the threat that the explicit career structure reinforces the "upstairs, downstairs" division between academics and technical contributors.

Additionally, such a change reduces the power that institutional logics from research have over decisions made in research software engineering, allowing practitioners to make situational decisions that are more aligned with their work on research software than with traditional models for research activity.

Goth et al. propose a three-tier career pathway from Junior RSE via Senior RSE to Principal RSE, with the Principal role varying depending on whether the RSE is more academic-focused or more technically-focused [88, §5.1]. Additionally, we saw some evidence in our data that there are existing examples of RSE career structures to learn from, with Participant 3 telling us that their department runs "our own hiring... our own regrading process, which means that we have control and understanding of the roles". Similar to the suggestion made by Goth et al., their structure bifurcates at higher levels depending on whether the individual pursues a more technical or managerial career.

Finally, while we discovered debates about professional ethics in software engineering, which we reported in section 2.3.5, we did not find evidence that RSEs appeal to a code of ethics in performing their work. Even though we asked interview participants about service to others (see section 6.3.3), the responses we received referred to the liminality of RSE in the academic space rather than to the occupation as a social or public good. We recommend that any attempt to formalise an education programme for RSE, or to license practitioners, should create and promulgate a code of ethics that helps engineers to balance the needs of their employers, research colleagues, participants and animal subjects (where relevant), and society. RPOs typically already provide policy and training on research integrity and research ethics, so are well-placed to implement a code of research software practice.

10.3.4 Researchers

Academics themselves bear some responsibility for the "service or subservience" relationship, as RSEs perceive their researcher colleagues to be "snobbish" or "sharp-elbowed", constructing a two-tier professional culture in which one is either an academic or "a working-class person who cleans the glassware" (Participant 13). Even if some or all academics do not intend to create or reinforce this distinction, there is deliberate work needed to dismantle the perception that academics treat other contributions to research as secondary.

Researchers can adopt the Contributor Role Taxonomy (CRediT) [232] to ensure that all participants in their research are appropriately acknowledged for their contributions. In many situations, a research software output may make equal contribution to the state of the art as a scholarly publication, and sometimes the software is the main output of a research endeavour [233]. Giving appropriate credit for software contributions increases the visibility and status of the contributors, and improves the sustainability of research software efforts [224]. The benefit of this action would extend beyond RSEs to a broad ecosystem of other research technical

professionals who contribute intellectually to research but who aren't recognised through traditional mechanisms for authorial credit on research papers [234].

Going beyond giving credit, researchers should involve contributors including RSEs in planning and ongoing review of research projects, to ensure that they have the resources and information they need to contribute to the project's success.

10.4 Further Work

This study represents a first attempt at providing substantive evidence of the practices and performance of research software engineering, and a theoretically informed analysis. There are many directions in which future research could take this work.

Our work represents a snapshot of the state of RSE after its first decade. Repetitions of this work would allow for longitudinal study of the development of an emerging profession within academia, exploring whether RSE emerges from the shadows of the research profession or remains liminal.

Building on our data by expanding the participant base while keeping the constructivist epistemology would enable future researchers to achieve saturation and generate a Charmaz-style grounded theory [165]. In this methodology, researchers build multiple levels of categorisation of participants' data to generate a theory with explaining power in the research topic. An explaining theory of RSE would enable leaders, practitioners, funders, and other stakeholders to make informed predictions about interventions to improve the state of research software.

Both a longitudinal study of RSE, and an expanded study that reaches saturation, would help to address the threats identified in sections 5.3.1, 6.4.3, 7.4.4, and 8.4.4, that more data (and, particularly, more diverse recruitment of participants) would provide richer results and stronger support for conclusions drawn from analysing this research.

There are multiple avenues we explored in this thesis that could form the basis of future work, by studying particular details in depth. For example, we found that RSE mostly supports scientific computing, and future work could investigate the

distribution of expertise between different disciplines or domains. This work could identify centres of research software excellence that would make good case studies for others to emulate. As another example, researchers could further explore the relationship between RSEs and people in traditional academic roles, characterised in this thesis as "service or subservience". This work could both study the RSE-researcher dynamic in more depth, and broaden the scope to consider research data scientists, librarians, research technical professionals, and other service-providing roles, to help institutions understand and improve their research culture (see section 2.4.2). By considering other roles, researchers could further understand the professional relationships that people in research form, identified in section 8.5.

One particular topic of interest could be to explore how "institutional memory" (see section 3.2.5) is applied by groups with different organisational structures, to understand how software knowledge becomes part of the institutional memory either of RSE groups, or of research groups that work with RSEs.

Another relevant area is the transition research makes between layers of the research software stack—seen briefly in FG1 where software created by researchers for specific papers goes on to be used by other researchers in the group and in the wider research-domain community. Similarly, using the model of research software that categorises the intent behind the software described in [223], researchers could investigate how software moves from being a one-off script or "professorware" to being a sustained research software product. Understanding how research software goes from single-user to multiple users would help to shape the checklist for sustainable software, and the research software requirements methodology, described in section 10.3. This topic would benefit from a similar mixed-methods approach to that applied in this thesis. Software publications, research publications that cite the software, source code repositories and issue trackers comprise quantitative data that could support the study; while interviews with researchers, software creators, funders, and other stakeholders, and focus groups examining how those people work with the software, could form the qualitative components.

10.5 Summary of Contributions

The contributions we make in this thesis include empirical contributions that advance our understanding of research software engineering, and of the state of computer science application in other research disciplines; theoretical contributions in selecting appropriate theoretical approaches to analyse our data, and combining results derived from multiple theories; methodological contributions in designing a mixed methods study for a field that has yet to be the subject of systematic, evidence-based investigation; and practical contributions in the form of recommendations for multiple stakeholders in RSE.

10.5.1 Empirical Contributions

The mixed methods study we present in this thesis provides an empirical "snapshot" of the state of RSE at the time of writing: as discussed in section 3.2.6, this is the first systematic investigation of RSE to include qualitative components. The answers to our research questions we present in section 10.2 provide an empirical basis on which to critique the opinion-based approaches to RSE that are found in the literature (see section 3.2.2).

As results in computer science are the scientific basis for software engineering, and as research software engineering is the application of software engineering to research, the snapshot we present in this thesis illuminates the extent of interdisciplinary application of computer science results in other fields, and the possibilities for greater interdisciplinary collaboration. For example, RSE is most prevalent in scientific fields, suggesting that their are still unexplored opportunities for co-operation between computer scientists and researchers in the arts, humanities, and social sciences.

10.5.2 Theoretical Contributions

In section 3.6 we review theoretical positions on the analysis of professions and professionalism. In section 4.3 we consider the relevance of these theories to our work in constructing our research methodology, selecting certain theoretical

approaches and excluding others. We use the following theories in analysing the qualitative strands of our work:

- The functionalist theory of professions, which we apply to our interview study in chapter 6.
- Neo-institutionalist theory, which we apply to our study of RSE blogs in chapter 7.
- Communities of practice, which we apply to our focus group sessions in chapter 8.

We combined the results from these chapters in chapter 9, relating the findings from the different strands as well as connecting data from each strand to the theoretical analyses presented in the other strands. We took this broad, multi-faceted approach to our research because RSE is not already the subject of systematic qualitative or mixed-methods investigation, so no one framework or theory had clear benefits or relevance. Nonetheless, we excluded Marxian and neo-Weberian theories, because these "beg the question" by assuming particular interactions or power relationships between people within the area of study.

We adopt a constructivist epistemology in this research, which we explain and justify in section 4.4.

Future researchers—investigating the development of RSE or of other professional fields—can use this theoretical basis to design their own research approaches. They can choose to adopt the same theoretical structure and epistemological foundation as that presented in this thesis, to develop ideas and results we present. Alternatively, they can take a contrasting position, adopting alternative theories, to critique the research presented here, or to discover whether alternative theoretical analysis yields similar results.

We suggest particular approaches to further work in section 10.4, above.

10.5.3 Methodological Contributions

Having discovered in chapter 3 that much of the literature on RSE comprises the opinions of practitioners on how to improve research software and doesn't appeal to an empirical basis, we provide a substantive, evidence-based approach to understanding the field. We apply multiple theoretical perspectives to our mixed methods study, to supply an informed analysis of the data that draws on multiple academic streams.

We combine data gathered in different contexts, presented by participants in different settings:

- The "one researcher, many (anonymous) respondents" basis of the survey in chapter 5;
- The "one researcher, one participant" basis of the interviews in chapter 6;
- The "one author, many (anonymous) readers" basis of blog posts gathered in chapter 7; and
- The "one researcher, few participants" basis of focus groups in chapter 8.

Applying all of these approaches means that we interact with participants in diverse ways (and, in the case of their blog posts, in an abstract way as a public member of their blog's audience) and therefore understand their perspectives in a richer way, taking into account different interactions between the researcher and participants, and the participants with their colleagues and with the field at large. We believe that our methodology would, in particular, form a good basis for a constructivist grounded theory development in RSE, following Charmaz [165].

10.5.4 Practical Contributions

We present practical recommendations based on the results of our research in section 10.3, above. In summary, they are:

- Funders of research should develop a checklist approach to identifying software projects that need to be sustained to support the research community.
- Funders should ensure that, for software that needs to be sustained, funding for ongoing sustaining work is costed into grants.
- RSE leaders should work with funders to identify skills gaps in each research domain.
- RSE organisations should develop "outreach" programs to formally connect researchers with RSE mentors to transfer software engineering knowledge.
- Institutions that hire RSEs should organise staff along agency or consultancy lines, rather than as "embedded" members of particular research groups, and define a career structure that parallels the research career but with expectations specific to RSE.
- Educators and professional bodies in RSE should develop an ethical code that their students and members adhere to.

We believe that adopting these recommendations would close the "chasm" in software engineering knowledge between research and industry, and provide a firmer basis for RSE to operate within research culture. The benefits of doing so would be to make research software more efficient and effective, research that depends on software more reliable, and to generate better impact for research software investment.

10.6 Personal Reflections

At the beginning of this research, I had an exceedingly positivist approach to the world, which had served me well through an undergraduate degree in Physics, a masters degree in software engineering, and a two-decade career in computing including Research Software Engineering. On becoming an RSE, I had been asked by incumbent peers and leaders to bring my "industry experience" to their research software problems. This led me to question which parts of my experience

were relevant, and indeed what this "industry" was that academics thought were doing a better job of creating software. Having worked for small startups, large multinationals, companies that operated in technology, finance, telecommunications, and other sectors, I had seen many different practices in pursuit of myriad goals: what of this was "industry software engineering", and what of it did RSE need?

Deeper than that, I wanted to know whether this was even an appropriate line of development: whether the practices that those in industry pursue produce outcomes that those in research would value. This was the question that led me to pursue this research project, a question that I rapidly modified to the research questions presented in section 4.2 as discussions with my supervisors revealed both the enormous scope of my initial question, and the amount of knowledge in software engineering (both industry and research flavours) that is taken for granted by practitioners and would need rigorous investigation before my initial questions could even lead to testable hypotheses.

As part of this process I discovered that my positivist toolset was only one of multiple epistemological approaches to answer questions about how people work, and as I explored the philosophical questions that arose, over whether there exists any objective truth to software engineering, and whether people collaborating in software products are uncovering shared reality or creating it, I realised that positivism wasn't going to be the best approach to investigating my questions.

Adopting a social-constructivist paradigm meant that I had to account for my contribution to the research, not merely as an observer trying to uncover the truth but as a participant in a conversation that builds a shared reality, and also as a practitioner of (research) software engineering with my own preconceptions of what works and how things "should" be done. To reflect on my own position in the research, I made use of the memoing techniques adopted by grounded theory researchers [164, pp. 52-55]. I wrote memos reflecting on the conversations I had in my interviews and focus groups, examining the paths I had led the conversations down, whether I had allowed or encouraged participants to fully present their thoughts, or guided them in any way. Included in these memos were reflections on

how the conversation connected or differed from both the other data that I had gathered, and the expectations or preconceptions I had. I also wrote memos that reflected on whether I found aspects of the conversations expected or surprising, based on my experiences, particularly writing down my thoughts on each of the blog posts that I analysed in chapter 7.

I created memos on reviewing interview and focus group transcripts, and on reading blog posts, so that I had an awareness of my own perspective as I engaged in initial coding and could account for that perspective in analysing the data. This is an example of a memo I created after one of the interviews:

> Got a very different perspective from this participant, as they are at a well-established organisation that has defined career pathways, project management systems, and has the autonomy to implement those things without having to fit into existing university structures. Also HPC so different type of project, a lot more helping other people who have software to use their facilities than doing the software development themselves, and more emphasis on support, maintenance, infrastructure etc.
>
> But interesting that along with participant 1, participant 3 identifies vague and changing requirements as the big problem despite having the formal project management process! I wonder whether this is the big deal: not software quality, but knowing what to build and communicating how the build process will work with the academics.

I also created memos between coding phases to help organise my thoughts on what categories had arisen, and what interesting or surprising ideas were presented in the codes. This example is an extract from a memo I wrote while organising codes:

> What is the model of interaction for RSEs? Both objectively (how is their work with collaborators organised) and subjectively (how is their perception of their relationship, seniority/superiority/etc). Try to keep objective and subjective analysis separate as far as possible.

> When people recommend practices to others, is that for practical reasons or evangelism? Are the practical reasons for themselves or for the other person's benefit?
>
> RSE as labels: who are the other labels? What status is conveyed by that label, and does everybody agree? Do "embedded" and "facility" RSEs use the same labels, perceive themselves the same way, perceive the role the same way? How does this agree/conflict with the RSE leadership view of the role?
>
> And ultimately what is the success of RSE? Is it that everybody knows the software engineering skills and there are then no RSEs, or is it that this career niche is carved out? This is central to the question of whether RSE is a profession.

Additionally, I wrote memos at any time that a relevant thought or question came to me, and the index in NVivo shows that this occasionally happened late in the evening or early in the morning when I wasn't deliberately working on the research. This short memo is an example of one that I briefly noted one evening when a thought occurred, and that I subsequently referred back to when considering how to categorise codes in the data:

> [I]t can be hard to decide whether something is a "tool" or a "practice", and whether that's a genuine distinction. Is version control a tool or a practice? If version control is a practice, is git a child code of version control or is it a tool?
>
> I suppose I need to study more how the participants use these words. I think github, git, and VC all seem to be used interchangeably, so maybe I won't end up needing a tools hierarchy. I'll have "what we're doing" -> "how we do it" -> "what we use" hierarchies instead.

Undertaking this project has broadened my understanding of research, particularly research involving human participants, and the complexities involved in exploring and understanding human interactions, even in the relatively limited domain of the software engineer's workplace. It has also opened up new avenues for organising and presenting this work.

Further, the research has deepened my understanding of software engineering, both as a practice I and others undertake and an identity that we adopt and participate in defining. I do not believe that I have come close to answering my earlier question of whether industry software engineering practices produce outcomes that research software engineers would value, but the work presented here does lay useful groundwork for developing answers to that question.

10.7 Final Thoughts

Research software engineering may be a technical discipline within academia, but it is also a social movement, aimed at changing the way that researchers engage with the software they need for their research, and its construction. Its practitioners participate out of interest and enjoyment—Participant 11 described their transition from researcher to software engineer as involving "a breaking point where I was just like No, I really don't like Gaussian processes. I wanna write Python libraries. I'm a software engineer." However, to make the change persist, practitioners need to promote and demonstrate the benefits to other stakeholders in research. In his pamphlet "What is to be Done?: Burning Questions of our Movement", V.I. Lenin posited that four roles are necessary for a revolution to take hold: theoreticians, propagandists, agitators, and organisers.

> The theoreticians write research works on tariff policy, with the "call", say, to struggle for commercial treaties and for Free Trade. The propagandist does the same thing in the periodical press, and the agitator in public speeches. At the present time [1901], the "concrete action" of the masses takes the form of signing petitions to the Reichstag against raising the corn duties. The call for this action comes indirectly from the theoreticians, the propagandists, and the agitators, and, directly, from the workers who take the petition lists to the factories and to private homes for the gathering of signatures. [235, Source: Marxists Internet Archive]

It is our hope that the work we present in this thesis acts as a theoretical basis for propagandists, agitators, and organisers to play their parts, revolutionising

research and allowing future generations to stand on the shoulders of the giants creating today's research software.

Appendices

A Interview Script

Contents

A.1 Introduction

This appendix contains the script used to guide interviews presented and discussed in chapter 6. As the interviews were open-ended, the questions shown here were used as the starting points for deeper discussions.

A.2 Recruitment Message

This message was distributed via the Society of RSE Slack chat, the society's newsletter, and by email to potential participants identified by theoretical sampling.

> I am recruiting participants for a Grounded Theory study into the profession of Research Software Engineering. This study will form my doctoral thesis, and the goal of the research is

to produce an evidence-based "manifesto" for RSE.

The first round of this research will comprise open-ended interviews with stakeholders in the field of research software. The interviews will be conducted via Microsoft Teams, and will be recorded for transcription and analysis (the transcripts and any data derived from them will be anonymised, and identifying information redacted). This work has received ethical approval from the University of Oxford Department of Computer Science ethics committee, identifier CS_C1A_021_043.

Participation is entirely voluntary. If you're interested, please send me a message on Slack or contact me via email graham.lee@cs.ox.ac.uk to schedule an interview. Thank you very much!

Graham.

A.3 Introduction and Oral Consent

This message was read to participants at the start of the interview, and their consent sought before recording started. When a participant gave their consent, we started recording, and asked for confirmation of their consent, so we had a record of their grant.

Hello, thank you for taking part in my doctoral research project on the professionalisation of Research Software Engineering. This interview will take about an hour, and will be open-ended: I'll ask some broad questions about RSE then you're driving the conversation, and we'll go where you take us. With your permission I'd like to record and transcribe this interview, please. The recording of this interview and unedited transcript will only be accessible by me, and anonymised transcripts may be made openly available as part of the data appendix for my thesis, and other papers that come from this research. You should be aware that as RSE is a fairly small field, other participants may be able to identify you even from anonymised statements, and you should bear this in mind during the interview. You can withdraw your consent at any time, and if you do so your data will be removed from the study. You can also

request a copy of your interview transcript at any time. Do either by sending me an email: graham.lee@cs.ox.ac.uk.

The discussion we have in this interview is intended purely to inform my research, and to have no bearing on our professional relationship. I welcome your openness and honesty in your answers, though I understand that there could be issues arising from discussing approaches to work with a colleague so bear in mind you can decline to answer any question, and also withdraw your consent for me to use your data in the research at any time. You can also ask to review the transcript of this interview at any time. If you are worried that the information you disclose in this interview is being used outside the scope of this research, please contact your line manager, the Computer Science HR department (hr@cs.ox.ac.uk), or the Departmental Research Ethics Committee (ethics@cs.ox.ac.uk).[1]

Are you willing to proceed with the interview, and do you give consent for the interview to be recorded?

A.4 Demographic questions

We asked these questions to learn about the participant, their background, and their role in the research software ecosystem.

1. What is your job title?
2. Where do you work, and what kind of organisation is it?
3. How does your role involve you in the creation of research software?
4. How is your work on research software funded?
5. How much experience do you have?
6. Do you have any roles in RSE community organisations?

[1] This paragraph was added in cases where the participant worked for the University of Oxford, and therefore had a working relationship with the author.

A.5 Questions on the Profession

We asked these questions as the basis of open-ended interview strands that address RQ2.

1. Does RSE have a clear body of knowledge that engineers are expected to understand? How do RSEs acquire and develop that knowledge?

2. Do RSEs have autonomy in their work? What other stakeholders, if any, have a say in how research software is made?

3. Do RSEs in general value service to others in their work? How is this value (or its absence) demonstrated?

4. Is being an RSE a privileged position at work or in society? What are RSEs and their leaders doing to develop or sustain the status of the role?

5. Does RSE have distinct pathways, progression, and expectations from other research occupations?

6. To what extent is RSE distinct from "industry software engineering" or a context-independent software engineering occupation?

7. What problems do RSEs commonly face in their research?

8. How do the people involved in research software engineering, whether RSEs or others, think about and measure the performance and capability of their software engineering work?[2]

A.6 Closing Statement

We read this statement to participants to thank them for their time, and to offer them the opportunity to join the participants' council for this study.

[2] As the research progressed we noted that we needed to clarify that this question is about the performance of the work being done, not about measuring the performance of research software.

Thank you for taking part! I'd like to invite you onto the participants' council for this project. The council meets twice per Oxford university term, and gives participants an opportunity to discuss the direction of the research, to identify unexplored areas, and to raise issues or concerns relating to the research and its application. The council meets online over videoconference, so were you to participate you would be identifiable by other council members. Would you like to join the council, and do I have your consent to add your email address to the members' mailing list? Thanks again!

B

Blogs on Research Software Engineering Included in Analysis

Contents

B.1 Introduction

This appendix lists the blogs on RSE that we discovered as source data for the analysis in Chapter 7, and the articles from each that were included in the analysis. Each subsection heading gives the title of a blog that our search discovered. Where we discovered a blog but did not use any content from it in the analysis, we give

our reasoning in the subsection; otherwise, we list the articles used.

B.2 List of Blogs and Articles

B.2.1 Software Sustainability Institute - Cultivating world-class research with software

Published at https://www.software.ac.uk/blog.

Blog 1. "Rescuing Linesman" https://software.ac.uk/blog/2023-04-28-rescuing-linesman

Blog 2. "Presenting at RSECon22 and starting an RSE Group in South Africa" https://software.ac.uk/blog/2023-04-27-presenting-rsecon22-and-starting-rse-group-south-africa

Blog 3. "Code for Thought: Who Are We?" https://software.ac.uk/news/code-thought-who-are-we[1]

Blog 4. "The Alan Turing Institute to sponsor Collaborations Workshop 2023" https://software.ac.uk/blog/2023-04-24-alan-turing-institute-sponsor-collaborations-workshop-2023

B.2.2 Research Software Engineering

Published at https://blogs.imperial.ac.uk/research-software-engineering/.

Blog 5. "Python Development on M1 Macs" https://blogs.imperial.ac.uk/research-software-engineering/2023/04/20/python-development-on-m1-macs/

B.2.3 RSE Sheffield Blog

Published at https://rse.shef.ac.uk/blog/. This blog did not offer an RSSfeed at time of retrieval.

[1]We note the reflexivity of this item; it announces a podcast episode involving the author that was motivated by discussions on the research in this thesis.

B.2.4 Thoughts and reflections from the Lab

Published at https://kdl.kcl.ac.uk/blog/. We did not find any posts within our target date range.

B.2.5 Invenia Blog

Published at https://invenia.github.io/blog/. We did not find any posts within our target date range.

B.2.6 Alan Turing Institute: Research Engineering Blogs and News

Published at https://www.turing.ac.uk/research/research-engineering/reg-blogs-and-news. This blog did not offer an RSSfeed at time of retrieval.

B.2.7 US-RSE Community Syndicated Blog

Published at https://us-rse.org/blog/. This blog did not offer an RSSfeed at time of retrieval.

B.2.8 OxRSE

Published at https://www.rse.ox.ac.uk/posts/. We did not find any posts within our target date range.

B.2.9 BSSw Blog

Published at https://bssw.io/blog_posts.

Blog 6. "Supporting Reproducibility and Replicability Initiatives" https://bssw.io/items/supporting-reproducibility-and-replicability-initiatives

Blog 7. "ISC23 Software-Related Events" https://bssw.io/items/isc23-software-related-events

Blog 8. "How Open Source Tooling Is Changing the Way Professional Researchers Learn to Code" https://bssw.io/items/how-open-source-tooling-is-changing-the-way-professional-researchers-learn-to-code

Blog 9. "Best Practices for HPC Software Developers Webinar Series" https://bssw.io/items/best-practices-for-hpc-software-developers-webinar-series

Blog 10. "What are Conferences and Workshops for Better Scientific Software?" https://bssw.io/items/what-are-conferences-and-workshops-for-better-scientific-software

Blog 11. "Julia's Value Proposition for Better Scientific Software" https://bssw.io/items/julia-s-value-proposition-for-better-scientific-software

Blog 12. "What Is a Research Software Engineer?" https://bssw.io/items/what-is-a-research-software-engineer

Blog 13. "Is Project Management Killing your Software Product?" https://bssw.io/items/is-project-management-killing-your-software-product

Blog 14. "What Is Online Learning?" https://bssw.io/items/what-is-online-learning

Blog 15. "What Is In-Person Learning?" https://bssw.io/items/what-is-in-person-learning

Blog 16. "A Collection of Open Source Software Maintainer Guides" https://bssw.io/items/a-collection-of-open-source-software-maintainer-guides

Blog 17. "Busy != Productive" https://bssw.io/items/busy-productive

C

Script used to enumerate RSE blog articles for analysis.

Contents

C.1 Introduction

This script, `rss-fetch.pl`, is written in the Perl 5 programming language. It takes as command-line arguments a start date, end date, and the URL for a blog's RSS feed. The script fetches and decodes the feed, then filters the entries for those published between the specified dates. It prints a list of the article titles, URLs, and dates of publication for matching articles. It additionally logs its progress to the standard error stream.

C.2 Dependencies

The script relies on the Perl 5 programming language, available at https://www.perl.org/get.html. We tested it with version 5.36.1 of Perl 5, and it should work with any newer version. Additionally, the script requires the following modules from the Comprehensive Perl Archive Network (CPAN):

- `DateTime`
- `GetOpt::Long`
- `HTTP::Request`
- `LWP::UserAgent`
- `Time::ParseDate`
- `XML::RSS::Parser`

C.3 Usage

Assuming the script is in your current working directory, invoke it with a command like this:

```
./rss-fetch.pl --feed https://society-rse.org/feed/ --from 2023-01-01 --to 2023-04-30
```

C.4 Source Code

```
#!/usr/bin/env perl -w

use strict;
use warnings;

use DateTime;
use Getopt::Long;
use Time::ParseDate;
```

```
use XML::RSS::Parser;
require HTTP::Request;
require LWP::UserAgent;

sub parse_date {
    my $string = shift;
    my ($year, $month, $day) = split(/-/, $string);
    return DateTime->new(year => $year,
                         month => $month,
                         day => $day);
}

my $feed = "";
my $from_date = "";
my $to_date = "";

GetOptions("feed=s" => \$feed,
           "from=s" => \$from_date,
           "to=s" => \$to_date)
    or die("Error in command-line arguments\n");

my $from = parse_date($from_date);
my $to = parse_date($to_date);

# Fetch the RSS feed.
my $req = HTTP::Request->new(GET => $feed);
my $ua = LWP::UserAgent->new;
my $response = $ua->request($req);

if ($response->is_success) {
```

```
    print(STDERR "Fetched the feed.\n");
    my $feed_content = $response->decoded_content;
    my $rss_parser = XML::RSS::Parser->new;
    my $feed = $rss_parser->parse_string($feed_content);
    my $feed_title = $feed->query('/channel/title')->text_content;
    print("Title: $feed_title\n");
    foreach my $item ($feed->query('//item')) {
        # If the item date is between from and to
        my $item_date = $item->query('pubDate')->text_content;
        my $item_epoch = parsedate($item_date, NO_RELATIVE => 1);
        my $published_date = DateTime->from_epoch($item_epoch);
        my $item_title = $item->query('title')->text_content;
        if ($published_date < $from) {
            print(STDERR  "Found \"$item_title\" but it was published too early.\n");
        } elsif ($published_date > $to) {
            print(STDERR  "Found \"$item_title\" but it was published too late.\n");
        } else {
            # Give me the URL
            my $link = $item->query('link')->text_content;
            print(" - \"$item_title\" published on $published_date\n");
            print("  - $link\n");
        }
    }

} else {
    print STDERR $response->status_line . "\n";
}
```

D

Focus Group Information Sheet

Contents

D.1 Introduction

This appendix contains the information sheet sent to potential recruits for the focus groups, discussed in chapter 8.

D.2 Participant Information Sheet

Central University Research Ethics Committee Approval Reference: **CS_C1A_24_001**

D.2.1 Introductory paragraph

You are being invited to take part in a research project. Before you decide it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. Take time to decide whether you wish to take part.

D.2.2 Why is this research being conducted?

We are researching how research groups work to create the research software they use. We want to understand the problems researchers find in creating and using custom software, and how research software engineering (RSE) can solve some of these problems and help researchers to be more effective.

D.2.3 Why have I been invited to take part?

You have been selected because your research group creates research software. Additionally, you are over 18.

D.2.4 Do I have to take part?

No. It is up to you to decide whether to take part. You can withdraw yourself from the research, without giving a reason, and without negative consequences, by advising me/ us of this decision. Alternatively, if you want to redact specific statements from the research, you can also advise me/us of this. The deadline by which you can withdraw any information you have contributed to the research is October 31, 2024. If you decide to withdraw or to redact certain information, we will remove it from the transcripts of your group's activities, along with instances where other participants use that information. We will not use the withdrawn information in our thesis.

D.2.5 What will happen to me if I take part in the research?

Members of your research group will be invited to take part in a focus group about how the group works to create research software. This will be an in-person activity, held in a meeting room in your University, and will take one hour. One of us (Graham Lee) will facilitate the group, by leading you through a couple of activities that investigate how your group collaborates to work on your research, and the software development work that forms part of the research. This is a single activity; you will not be asked to do any ongoing work or to join in another meeting.

With your consent, we would like to video record you, because we want to study not only the outcomes of the focus group activities but the conversations among your group members as you take part. Consent will be taken via a written form that you complete before the focus group begins; you can pause or stop any time and you can withdraw consent or ask that particular information (for example, statements you made) be redacted at any time before 31st October 2024.

D.2.6 What are the possible disadvantages and risks in taking part?

You will be discussing your work with other members of your research group, so it's possible that any workplace politics or negative aspects of relationships with your colleagues could come out as part of the discussion. We request that you engage collegially with your teammates, and ask you to avoid any sensitive topics including personal categories like race, gender, and trade union membership.

Your data will be anonymised by assigning numeric identifiers to the individuals, groups, and institutions involved. Any data published as part of Graham Lee's thesis will use the anonymised version, so no participant will be directly identifiable. However, as Research Software Engineering is a small and well-connected field, it's possible that some people could identify others indirectly even from anonymised data.

D.2.7 Are there any benefits in taking part?

While there are no immediate benefits for those people participating in the research, it is hoped that this research will lead to a better understanding of how Research Software Engineering can lead to better outcomes for research groups that rely on software.

D.2.8 What information will be collected and why is the collection of this information relevant for achieving the research objectives?

I am interested in your experiences of developing research software, either as a researcher who uses the software or as a programmer creating it, or both. The information you provide will help me better understand the relationship between research and research software engineering, to answer my research question on how researchers and RSEs organise their work to create research software.

The researcher and his supervisors (Professor David Gavaghan, Professor Susanna-Assunta Sansone, and Dr. Helena Webb) will have access to the research data. Identifiable data (including consent forms) will be stored on the University of Oxford Nexus365 OneDrive service. Other research data will be stored for 3 years after publication or public release of the work of the research. The identifiable data, including consent forms, will not be shared with anyone other than the researcher and his supervisors.

D.2.9 Will the research be published? Could I be identified from any publications or other research outputs?

The findings from the research will/may be written up in my thesis. Your data will be anonymised by assigning numeric identifiers to the individuals, groups, and institutions involved. Any data published as part of the thesis will use the anonymised version, so no participant will be directly identifiable. I would like your permission to use direct quotations, but without directly quoting you, in any research outputs.

A copy of my thesis/dissertation will be deposited both in print and online in the Oxford University Research Archive where it will be publicly available to facilitate its use in future research.

D.2.10 Data Protection

The University of Oxford is the data controller with respect to your personal data, and as such will determine how your personal data is used in the research. The University will process your personal data for the purpose of the research outlined above. Research is a task that is performed in the public interest. Further information about your rights with respect to your personal data is available from the University's Information Compliance web site at https://compliance.admin.ox.ac.uk/individual-rights.

D.2.11 Who has reviewed this research?

This research has received ethics approval from a subcommittee of the University of Oxford Central University Research Ethics Committee. (Ethics reference: CS_C1A_24_001).

If you have a concern about any aspect of this research, please contact Graham Lee (graham.lee@cs.ox.ac.uk) or Prof. David Gavaghan (david.gavaghan@cs.ox.ac.uk) and we will do our best to answer your query. We will acknowledge your concern within 10 working days and give you an indication of how it will be dealt with. If you remain unhappy or wish to make a formal complaint, please contact the Chair of the Research Ethics Committee at the University of Oxford who will seek to resolve the matter as soon as possible:

The Chair, Computer Science Departmental Research Ethics Committee
Email: ethics@cs.ox.ac.uk. Address: Department of Computer Science, Wolfson Building, Parks Road, Oxford OX1 3QD.

D.2.12 Further Information and Contact Details

If you would like to discuss the research with someone beforehand (or if you have questions afterwards), please contact:

Graham Lee

Department of Computer Science

Wolfson Building

Parks Road

Oxford

OX1 3QD

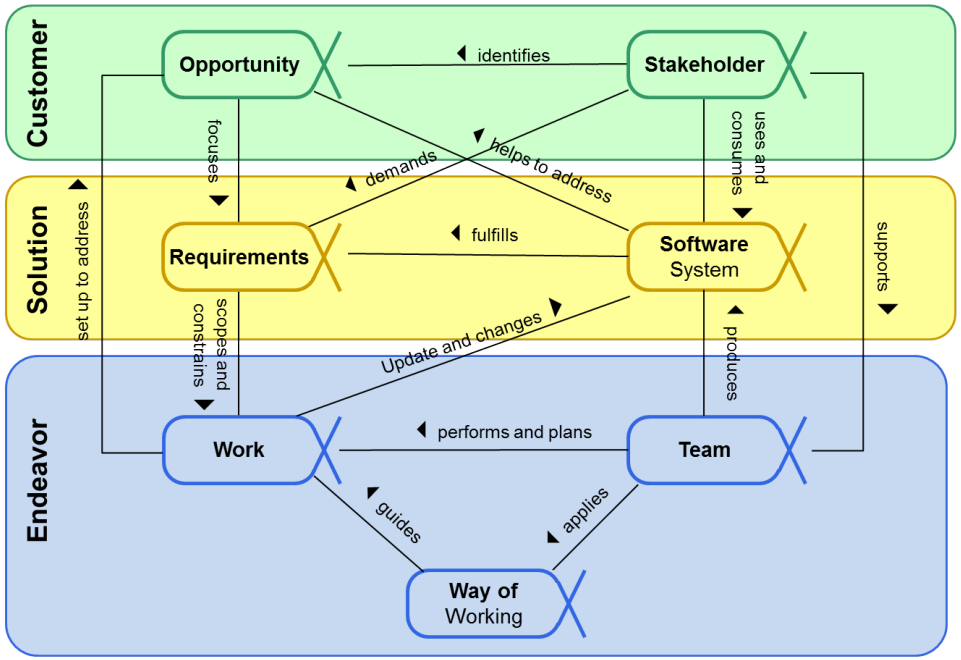
University direct line: 01865 210671

University email: graham.lee@cs.ox.ac.uk

E

Attributes of a Software Engineering Endeavour Included in the SEMAT Essence Alphas

Software Engineering Kernel



Opportunity

- Identified
- Solution Needed
- Value Established
- Viable
- Addressed
- Benefit Accrued

The set of circumstances that makes it appropriate to develop or change a software system.

The opportunity:

- Unites and motivates the stakeholders
- Has tangible benefit for the stakeholders
- Provides the justification for the system's development
- Establishes the value of the proposed system
- Represents the team's shared understanding of the stakeholder's needs

Stakeholders

- Recognized
- Represented
- Involved
- In Agreement
- Satisfied for Deployment
- Satisfied in Use

The people, groups, or organizations who affect or are affected by a software system.

The stakeholders:

- Provide the opportunity and are the source of the requirements
- Use and consume the software system
- Fund the development of the software system
- Actively represent the groups and organizations affected by the software system
- Are actively involved all the way through the endeavor
- Have representatives that collaborate with the team to reach agreement on an acceptable system

Requirements

- Conceived
- Bounded
- Coherent
- Acceptable
- Addressed
- Fulfilled

What the software system must do to address the opportunity and satisfy the stakeholders.

The requirements:

- Establish a shared understanding of what the software system must do
- Communicate the intent of the software system to be produced
- Define the capabilities, services and qualities that the stakeholders desire from the system
- Are organized to allow the scope of the software system to be managed
- Drive the development and testing of the system

Software System

- Architecture Selected
- Demonstrable
- Useable
- Ready
- Operational
- Retired

A system made up of software, hardware, and data that provides its primary value by the execution of the software.

The software system:

- Is the primary product of any software engineering endeavor
- Is structured, designed and implemented to fulfil the requirements
- Is architected to be maintainable, extensible and testable
- Provides value to its users and other stakeholders
- Should be kept bug free and easy to use
- Can be part of a larger software, hardware or business solution

Team

- Seeded
- Formed
- Collaborating
- Performing
- Adjourned

The group of people actively engaged in the development, maintenance, delivery or support of a specific software system.

The team:

- Is formed to complete a mission
- Is made up of one or more team members who collaborate together to complete their shared mission
- Is responsible for completing its work to an acceptable standard
- Supports its stakeholders in exploiting the opportunities and addressing the requirements
- Owns and continually improves its way-of-working

Work

- Initiated
- Prepared
- Started
- Under Control
- Concluded
- Closed

Activity involving mental or physical effort done in order to achieve a result.

The work:

- Is everything that the team does to produce a software system
- Is planned and performed by the team
- Is guided by the practices that make up the team's way-of-working
- Is sizeable, estimable and track-able
- Is broken up to minimize dependencies and reduce risk

Way of Working

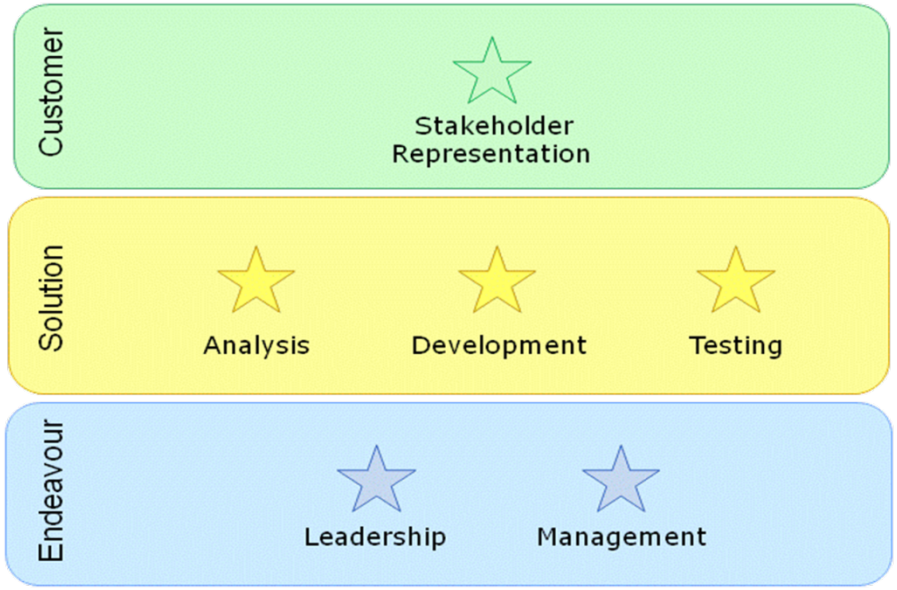
- Principles Established
- Foundation Established
- In Use
- In Place
- Working Well
- Retired

The tailored set of practices and tools used by the team members to guide and support their work.

The way of working:

- Helps the team be effective and work well together
- Evolves as the team understands its mission and environment, and undertakes its work
- Is continually inspected, adapted and improved
- Is owned and agreed by the team
- Reflects organizational policies and standards
- Reduces risk and helps to eliminate waste

Competency Overview



Stakeholder Representation

Innovates	5
Adapts	4
Masters	3
Applies	2
Assists	1

The ability to gather, communicate and balance the needs of other stakeholders, and accurately represent their views.

People with this competency help:

- The team to understand:
  - the business opportunity
  - the complexity and needs of the customers, users and other stakeholders
  - how well the system produced addresses the stakeholders' needs
- Negotiate and prioritize the requirements
- Interact with the stakeholders and developers about the solution to be developed

Analysis

Innovates	5
Adapts	4
Masters	3
Applies	2
Assists	1

The ability to understand opportunities and their related stakeholder needs, and transform them into an agreed and consistent set of requirements.

People with this competency help:

- Identify and understand needs and opportunities
- Identify the root causes of problems
- Capture, understand and communicate requirements
- Create and agree on specifications and models
- Visualize solutions and understand their impact

Development

Innovates	5
Adapts	4
Masters	3
Applies	2
Assists	1

The ability to design and program effective software systems following the standards and norms agreed by the team.

People with this competency help:

- Design and code software systems
- Formulate and/or evaluate strategies for choosing an appropriate design pattern or for combining design patters
- Design and leverage technical solutions
- Troubleshoot and resolve coding problems

Testing

The ability to test a system, verifying that it is usable and that it meets the requirements.

People with this competency help:

- Validate that the requirements and user needs are met
- Test the system
- Create the correct tests to efficiently verify the requirements
- Decide what, when and how to test
- Find defects and understand the quality of the system produced

Leadership

The ability to inspire and motivate a group of people to achieve a successful conclusion to their work and to meet their objectives.

People with this competency help:

- Resolve conflicts
- Inspire people to do their work
- Make sure that all team members are effective in their assignments
- Make and meet commitments
- Resolve any impediments or issues holding up the team's work
- The team to interact with stakeholders to shape priorities, report progress and respond to challenges

Management

The ability to coordinate, plan and track the work done by a team.

People with this competency help:

- Plan and coordinate activities over a defined period of time
- Track work completed and compare to plan
- Replan the work if necessary
- Proactively manage risks
- Account for time and money spent
- Interact with stakeholders to report progress

Bibliography

[1] D. Emmerson, "Breakout: Career track for software developers," Mar. 2012. [Online]. Available: https://groups.google.com/g/collabw12/c/xSdC0uz-IqA/m/8qhgrfceJKYJ

[2] S. Hettrick, "A not-so-brief history of Research Software Engineers," Aug. 2016. [Online]. Available: https://www.software.ac.uk/blog/2016-08-17-not-so-brief-history-research-software-engineers-0

[3] S. Hettrick, M. Antonioletti, L. Carr, N. Chue Hong, S. Crouch, D. De Roure, I. Emsley, C. Goble, A. Hay, D. Inupakutika, M. Jackson, A. Nenadic, T. Parkinson, M. I. Parsons, A. Pawlik, G. Peru, A. Proeme, J. Robinson, and S. Sufi, "Uk Research Software Survey 2014," Dec. 2014.

[4] M. Gruenpeter, D. S. Katz, A.-L. Lamprecht, T. Honeyman, D. Garijo, A. Struck, A. Niehues, P. A. Martinez, L. J. Castro, T. Rabemanantsoa, N. P. Chue Hong, C. Martinez-Ortiz, L. Sesink, M. Liffers, A. C. Fouilloux, C. Erdmann, S. Peroni, P. Martinez Lavanchy, I. Todorov, and M. Sinha, "Defining Research Software: A controversial discussion," Zenodo, Tech. Rep., Sep. 2021.

[5] K. Hinsen, "Dealing With Software Collapse," *Computing in Science & Engineering*, vol. 21, no. 3, pp. 104–108, May 2019.

[6] R. van Nieuwpoort and D. S. Katz, "Defining the roles of research software," *Upstream*, Mar. 2023.

[7] Y. Yehudi, M. Cashman, M. Felderer, M. Goedicke, W. Hasselbring, D. S. Katz, F. Löffler, S. Müller, and B. Rumpe, "Towards Defining Lifecycles and Categories of Research Software," Oct. 2024.

[8] R. Gardler, "Software Sustainability Maturity Model," http://oss-watch.ac.uk/resources/ssmm, Sep. 2010.

[9] M. Sako, "How generative AI fits into knowledge work," *Communications of the ACM*, vol. 67, no. 4, 2024.

[10] L. Banh, F. Holldack, and G. Strobel, "Copiloting the future: How generative AI transforms Software Engineering," *Information and Software Technology*, vol. 183, p. 107751, Jul. 2025.

[11] M. Simaremare, Triando, and S. Rico, "Exploring the Potential of Generative AI: Use Cases in Software Startups," in *Agile Processes in Software Engineering and Extreme Programming – Workshops*, L. Marchesi, A. Goldman, M. I. Lunesu, A. Przybyłek, A. Aguiar, L. Morgan, X. Wang, and A. Pinna, Eds. Cham: Springer Nature Switzerland, 2025, pp. 3–11.

[12] M. Ben Tayeb, V. Tikhonchuk, and J.-L. Feugeas, "ICF target optimization using generative AI," *Physics of Plasmas*, vol. 31, no. 10, p. 103903, Oct. 2024.

[13] M. Elliott, M. Luciano, and J. Fortes, "Integrating Large Language Models and the iDigBio Portal for Conversational Data Exploration and Retrieval," *Biodiversity Information Science and Standards*, vol. 8, p. e142696, Nov. 2024.

[14] M. A. Aziz, A. El-Zeiny, F. M. Hassan, and D. M. Naguib, "Coastal water quality dynamics of the Red Sea, southeast coast of Egypt using GeoAI and ChatGPT," *Journal of African Earth Sciences*, vol. 219, p. 105409, Nov. 2024.

[15] R. A. J. Borst, M. O. Kok, A. J. O'Shea, S. Pokhrel, T. H. Jones, and A. Boaz, "Envisioning and shaping translation of knowledge into action: A comparative case-study of stakeholder engagement in the development of a European tobacco control tool," *Health Policy*, vol. 123, no. 10, pp. 917–923, Oct. 2019.

[16] R. M. McClure, "Software Engineering: Report on a conference sponsored by the NATO Science Committee," in *Software Engineering*, Garmisch, Germany, Oct. 1968, p. 136.

[17] C. Jones, "Emergence of the Software Engineer," in *The Technical and Social History of Software Engineering*. Pearson Education, Inc., 2014, pp. 103–104.

[18] R. L. Glass, "The Standish Report: Does It Really Describe a Software Crisis?" *Communications of the ACM*, vol. 49, no. 8, pp. 15–16, Aug. 2006. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2102/10.1145/1145287.1145301

[19] H. Krasner, "The Cost of Poor Software Quality in the US: A 2020 Report," Center for Information and Software Quality, Tech. Rep., Jan. 2021. [Online]. Available: https://www.it-cisq.org/pdf/CPSQ-2020-report.pdf

[20] D. de Champeaux, "Software Engineering Considered Harmful," *Communications of the ACM*, vol. 45, no. 11, pp. 102–104, Nov. 2002. [Online]. Available: https://doi.org/10.1145/581571.581608

[21] B. J. Cox, "What if there's a Silver Bullet... And the Competition Gets it First?" *Journal of Object Oriented Programming*, vol. 5, no. 3, p. 7, Jun. 1992.

[22] C. Torrecilla-Salinas, J. Sedeño, M. Escalona, and M. Mejías, "Agile, Web Engineering and Capability Maturity Model Integration: A systematic literature review." *Information and Software Technology*, vol. 71, pp. 92–107, Mar. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S095058491500186X

[23] W. W. Gibbs, "Software's Chronic Crisis," *Scientific American*, vol. 271, no. 3, pp. 86–95, 1994. [Online]. Available: http://www.jstor.org/stable/24942840

[24] N. L. Ensmenger, "Conclusions: Visible Technicians," in *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. Cambridge, Mass., USA: MIT Press, 2021.

[25] B. Meyer, "Deconstructing agile texts," in *Agile!: The Good, the Hype and the Ugly*. Zurich: Springer International Publishing, 2014, pp. 17–30.

[26] T. Haigh, "'Crisis, What Crisis?' Reconsidering the Software Crisis of the 1960s and the Origins of Software Engineering," in *2nd Inventing Europe/Tensions Of Europe Conference*, Sofia, Jun. 2010.

[27] E. W. Dijkstra, "The humble programmer," *Commun. ACM*, vol. 15, no. 10, pp. 859–866, Oct. 1972.

[28] P. Hunter, "The reproducibility "crisis": Reaction to replication crisis should not stifle innovation," *EMBO Reports*, vol. 18, no. 9, p. 1493, Aug. 2017.

[29] British Computer Society, "IT certifications for professionals - BCS IT qualifications | BCS." [Online]. Available: https://www.bcs.org/qualifications-and-certifications/certifications-for-professionals/

[30] University of Oxford Software Engineering Programme, "University of Oxford Courses in Software Engineering—Subjects." [Online]. Available: http://www.cs.ox.ac.uk/softeng/courses/subjects.html

[31] Swansea University, "Software Engineering, BSc (Hons)." [Online]. Available: https://www.swansea.ac.uk/undergraduate/courses/science/computer-science/bsc-software-engineering-g600/

[32] T. L. Adams, "Interprofessional Relations and the Emergence of a New Profession: Software Engineering in the United States, United Kingdom, and Canada," *The Sociological Quarterly*, vol. 48, no. 3, pp. 507–532, 2007. [Online]. Available: http://ezproxy-prd.bodleian.ox.ac.uk:2154/stable/40220035

[33] P. Bourque, R. Dupuis, A. Abran, J. W. Moore, and L. Tripp, "The guide to the software engineering body of knowledge," *IEEE Software*, vol. 16, no. 6, pp. 35–44, 1999.

[34] J. Knight, N. Leveson, M. DeWalt, L. Elliot, C. Kaner, and H. Nissenbaum, "On Licensing of Software Engineers Working on Safety-Critical Software: Final Report of an ACM Task Force," Association of Computing Machinery, Tech. Rep., Oct. 2001.

[35] D. J. Frailey, "Licensing software engineers," *Communications of the ACM*, vol. 42, no. 12, pp. 29–30, Dec. 1999. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2102/10.1145/322796.322804

[36] J. C. Knight and N. G. Leveson, "Should software engineers be licensed?" *Communications of the ACM*, vol. 45, no. 2, Nov. 2002.

[37] P. Kruchten, "Licensing software engineers?" *IEEE Software*, vol. 25, no. 6, pp. 35–37, 2008.

[38] P. A. Laplante, "Licensing Professional Software Engineers: Seize the Opportunity," *Communications of the ACM*, vol. 57, no. 7, p. 3, Jul. 2014.

[39] G. McCalla, "Software engineering requires individual professionalism," *Communications of the ACM*, vol. 45, no. 11, pp. 98–101, Nov. 2002. [Online]. Available: https://doi.org/10.1145/581571.581606

[40] V. G. Cerf, "On Expert Testimony," *Communications of the ACM*, vol. 66, no. 12, p. 5, Dec. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3631277

[41] M. Y. Vardi, "Computing, You Have Blood on Your Hands!" *Communications of the ACM*, vol. 67, no. 1, p. 5, Dec. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3632963

[42] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development." [Online]. Available: https://agilemanifesto.org

[43] D. Wells, "Extreme Programming Rules," 1999. [Online]. Available: http://www.extremeprogramming.org/rules.html

[44] I. Jacobson, H. Lawson, P.-W. Ng, P. McMahon, and M. Goedicke, *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!*, 1st ed., ser. ACM Books. Morgan & Claypool, 2019, no. 25.

[45] P. McBreen, *Software Craftsmanship: The New Imperative*. Addison-Wesley, 2002.

[46] R. C. Martin, "Manifesto for Software Craftsmanship: Raising the Bar," 2009. [Online]. Available: https://manifesto.softwarecraftsmanship.org

[47] ——, "Manifesto Signatories," Aug. 2021. [Online]. Available: http://manifesto.softwarecraftsmanship.org/metrics

[48] K. Beck, "Test && commit || revert," Sep. 2018. [Online]. Available: https://medium.com/@kentbeck_7670/test-commit-revert-870bbd756864

[49] F. B. Aydemir and F. Dalpiaz, "A Roadmap for Ethics-Aware Software Engineering," in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, 2018, pp. 15–21.

[50] E. Iwata, "Former Octel engineer guilty of inside trading," *SFGate*, Aug. 1997. [Online]. Available: https://www.sfgate.com/business/article/Former-Octel-engineer-guilty-of-inside-trading-3104870.php

[51] M. Ritson, "Mark Ritson: Google memo reveals the many perils of gender discrimination," *Marketing Week (Online)*, Aug. 2017. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2082/magazines/mark-ritson-google-memo-reveals-many-perils/docview/1927038185/se-2?accountid=13042

[52] L. Dearden, "UK defence worker leaked top secret details of missile system, trial hears," *Independent*, Oct. 2020. [Online]. Available: https://www.independent.co.uk/news/uk/crime/uk-missile-system-leak-trial-defence-worker-simon-finch-mp-b1374856.html

[53] E. Pennink and B. Wright, "Disgruntled Ministry of Defence worker Simon Finch jailed after leaking top secret details about UK missile system," *Wales Online*, Nov. 2020. [Online]. Available: https://www.walesonline.co.uk/news/wales-news/official-secrets-act-breached-jail-19256396

[54] G. Lee, "Is there a "software engineering ethics"? Comparing commercial and research software engineering," in *2021 IEEE/ACM 2nd International Workshop on Ethics in Software Engineering Research and Practice (SEthics)*, 2021-06-04/2021-06-04, pp. 13–17.

[55] IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices, "Software engineering code of ethics and professional practice," *Science and Engineering Ethics*, vol. 7, no. 2, p. 8, 2001.

[56] N. G. Leveson and C. S. Turner, "An investigation of the Therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[57] T. Wang, M. Jerrett, P. Sinsheimer, and Y. Zhu, "Estimating PM2.5-associated mortality increase in California due to the Volkswagen emission control defeat device," *Atmospheric Environment*, vol. 144, pp. 168–174, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S135223101630680X

[58] H. Tuttle, "Volkswagen Rocked by Emissions Fraud Scandal," *Risk Management*, vol. 62, no. 10, pp. 4–7, Dec. 2015. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2082/scholarly-journals/volkswagen-rocked-emissions-fraud-scandal/docview/1747315487/se-2?accountid=13042

[59] Universities UK, "The concordat for Research Integrity," Oct. 2019. [Online]. Available: https://www.universitiesuk.ac.uk/policy-and-analysis/reports/Pages/the-concordat-for-research-integrity.aspx

[60] K. Kawai, "Design Concept of an Interactive Interface for Plant Operational Safety," *15th IFAC World Congress*, vol. 35, no. 1, pp. 377–380, Jan. 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015398761

[61] C. R. Hooper, T. Jivram, S. Law, A. Michell, and A. Somasunderam, "Using virtual patients to teach medical ethics, medical law and medical professionalism," *Medical Teacher*, vol. 34, no. 8, pp. 674–675, Aug. 2012. [Online]. Available: https://doi.org/10.3109/0142159X.2012.689450

[62] D. Weber-Wulff, "Plagiarism detectors are a crutch, and a problem," *Nature*, vol. 567, p. 435, Mar. 2019.

[63] M. U. G. Kraemer, S. V. Scarpino, V. Marivate, B. Gutierrez, B. Xu, G. Lee, J. B. Hawkins, C. Rivers, D. M. Pigott, R. Katz, and J. S. Brownstein, "Data curation during a pandemic and lessons learned from COVID-19," *Nature Computational Science*, vol. 1, no. 1, pp. 9–10, Jan. 2021. [Online]. Available: http://www.nature.com/articles/s43588-020-00015-6

[64] S. Russell and P. Norvig, "The Ethics and Risks of Developing Artificial Intelligence," in *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, Inc., 2016, pp. 1034–1040.

[65] T. Simonite, "When it Comes to Gorillas, Google Photos Remains Blind," *Wired*, Nov. 2018. [Online]. Available: https://www.wired.com/story/when-it-comes-to-gorillas-google-photos-remains-blind/

[66] J. Dastin, "Amazon scraps secret AI recruiting tool that showed bias against women," *Reuters*, Oct. 2018. [Online]. Available: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G

[67] I. Vesper, "Machine-learning algorithm quantifies gender bias in astronomy," *Nature News*, Nov. 2016. [Online]. Available: https://doi.org/10.1038/nature.2016.20932

[68] D. Leggett, "NHTSA to investigate Tesla autopilot fatality," *just-auto global news*, Jul. 2016. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2082/wire-feeds/nhtsa-investigate-tesla-autopilot-fatality/docview/1801318088/se-2?accountid=13042

[69] D. Etherington, "NHTSA's full final investigation into Tesla's Autopilot shows 40% crash rate reduction," *Techcrunch*, Jan. 2017. [Online]. Available: https://techcrunch.com/2017/01/19/nhtsas-full-final-investigation-into-teslas-autopilot-shows-40-crash-rate-reduction/?guccounter=1&guce_referrer=aHR0cHM6Ly9kdWNrZHVja2dvLmNvbS8&guce_referrer_sig=AQAAACO4cz0O9-HHqaf5FstgY5YvP8U5xWE8Eq-3oAQUf-VDnQeGkflbGKPdT28OmCHbAchsrunyPGG5OqGbB6hOs8AGpHX4R27_MUnmj0P7S-ki49Q3CRC70mbqPkrOtm0zTd25IHzQjhT8H5XtMTwvQgeH3JdILEPZXG0XsEj-yrSl

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. ukasz Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[71] Y. He, J. Chen, H. Dong, and I. Horrocks, "Exploring large language models for ontology alignment," in *ISWC 2023 Posters and Demos*. Athens, Greece: CEUR Workshop Proceedings, 2023.

[72] L. Daryanani, "How to Jailbreak ChatGPT," Feb. 2023.

[73] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '23. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 79–90.

[74] H. Nguyen, Z. He, S. A. Gandre, U. Pasupulety, S. K. Shivakumar, and K. Lerman, "Smoothing Out Hallucinations: Mitigating LLM Hallucination with Smoothed Knowledge Distillation," Feb. 2025.

[75] F. Ogunde, "Much ado about hallucinations: A brief assessment of the judicial response to large language model (LLM) hallucinations in the United States and Canada," *International Journal of the Legal Profession*, vol. 31, no. 2, pp. 187–205, May 2024.

[76] D. Kundaliya, "House of Lords Committee slams government inaction on LLM copyright misuse," https://www.computing.co.uk/news/4205153/house-lords-committee-slams-government-inaction-llm-copyright-misuse, May 2024.

[77] P. Glynn, "Artists release silent album in protest against AI using their work," https://www.bbc.com/news/articles/cwyd3r62kp5o, Feb. 2025.

[78] A. Birhane, A. Kasirzadeh, D. Leslie, and S. Wachter, "Science in the age of large language models," *Nature Reviews Physics*, vol. 5, no. 5, pp. 277–280, May 2023.

[79] S. Hettrick, "It takes more than publications to recognise everyone in research," Feb. 2020. [Online]. Available: https://www.software.ac.uk/blog/2020-02-19-it-takes-more-publications-recognise-everyone-research

[80] Software Sustainability Institute, "About the Software Sustainability Institute | Software Sustainability Institute," 2024. [Online]. Available: https://www.software.ac.uk/about

[81] ——, "Our funders | Software Sustainability Institute," 2024. [Online]. Available: https://www.software.ac.uk/about/our-funders

[82] UK Research and Innovation, "2022-23—2024-25 Budget Allocations for UK Research and Innovation," Jun. 2022. [Online]. Available: https://www.ukri.org/wp-content/uploads/2022/06/UKRI-241023-BudgetAllocationExplainer2022To2025.pdf

[83] Society of Research Software Engineering, "About." [Online]. Available: https://society-rse.org/about/

[84] N. Ferguson, D. Laydon, G. Nedjati Gilani, N. Imai, K. Ainslie, M. Baguelin, S. Bhatia, A. Boonyasiri, ZULMA. Cucunuba Perez, G. Cuomo-Dannenburg, A. Dighe, I. Dorigatti, H. Fu, K. Gaythorpe, W. Green, A. Hamlet, W. Hinsley, L. Okell, S. Van Elsland, H. Thompson, R. Verity, E. Volz, H. Wang, Y. Wang, P. Walker, P. Winskill, C. Whittaker, C. Donnelly, S. Riley, and A. Ghani, "Report 9: Impact of non-pharmaceutical interventions (NPIs) to reduce COVID19 mortality and healthcare demand," Imperial College London, Tech. Rep., Mar. 2020. [Online]. Available: http://spiral.imperial.ac.uk/handle/10044/1/77482

[85] M. S. Jalali, C. DiGennaro, and D. Sridhar, "Transparency assessment of COVID-19 models," *The Lancet. Global health*, vol. 8, no. 12, pp. e1459–e1460, Dec. 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/33125915

[86] UK Research and Innovation, "Early decisions made for REF 2028," Jun. 2023. [Online]. Available: https://www.ukri.org/news/early-decisions-made-for-ref-2028/

[87] A.-L. Lamprecht, C. Martinez-Ortiz, M. Barker, S. L. Bartholomew, J. Barton, N. C. Hong, J. Cohen, S. Druskat, J. Forest, J.-N. Grad, D. S. Katz, R. Richardson, R. Rosca, D. Schulte, A. Struck, and M. Weinzierl, "What Do We (Not) Know About Research Software Engineering?" *Journal of Open Research Software*, vol. 10, no. 1, p. 11, Dec. 2022.

[88] F. Goth, R. Alves, M. Braun, L. J. Castro, G. Chourdakis, S. Christ, J. Cohen, S. Druskat, F. Erxleben, J.-N. Grad, M. Hagdorn, T. Hodges, G. Juckeland, D. Kempf, A.-L. Lamprecht, J. Linxweiler, F. Löffler, M. Martone, M. Schwarzmeier, H. Seibold, J. P. Thiele, H. von Waldow, and S. Wittke, "Foundational Competencies and Responsibilities of a Research Software Engineer," Nov. 2023. [Online]. Available: https://arxiv.org/abs/2311.11457v3

[89] Y. V. Grossmann, G. Lanza, K. Biernacka, T. Hasler, and K. Helbig, "Software Management Plans – Current Concepts, Tools, and Application," *Data Science Journal*, vol. 23, no. 1, Sep. 2024.

[90] Science and Technology Facilities Council, "SCD CoSeC - Computational Science Centre for Research Communities." [Online]. Available: https://www.scd.stfc.ac.uk/Pages/CoSeC.aspx

[91] Society of Research Software Engineering, "Registration and Programme - RSECon25," https://rsecon25.society-rse.org/, Jun. 2025.

[92] P. Schmidt, "Code for Thought," https://www.buzzsprout.com/1326658, Jun. 2025.

[93] Oxford RSE Group, "Training | Oxford Research Software Engineering," https://www.rse.ox.ac.uk/training, 2025.

[94] D. S. Katz, K. McHenry, and J. S. Lee, "Research Software Sustainability: Lessons Learned at NCSA," in *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, pp. 7249–7256.

[95] Research Software Alliance, "Amsterdam Declaration on Funding Research Software Sustainability (1.1)," 2024.

[96] ——, "Signatories — ADORE.software."

[97] M. Jirotka, B. Grimpe, B. Stahl, G. Eden, and M. Hartswood, "Responsible research and innovation in the digital age," *Communications of the ACM*, vol. 60, no. 5, pp. 62–68, Apr. 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3064940

[98] B. C. Stahl, G. Eden, M. Jirotka, and M. Coeckelbergh, "From computer ethics to responsible research and innovation in ICT," *Information & Management*, vol. 51, no. 6, pp. 810–818, Sep. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037872061400007X

[99] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. d. S. Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, pp. 1–9, Mar. 2016. [Online]. Available: https://www.nature.com/articles/sdata201618

[100] S.-A. Sansone, P. McQuilton, P. Rocca-Serra, A. Gonzalez-Beltran, M. Izzo, A. L. Lister, and M. Thurston, "FAIRsharing as a community approach to standards, repositories and policies," *Nature Biotechnology*, vol. 37, no. 4, pp. 358–367, Apr. 2019. [Online]. Available: https://www.nature.com/articles/s41587-019-0080-8

[101] SpringerNature, "Reporting standards and availability of data, materials, code and protocols." [Online]. Available: https://www.nature.com/nature-portfolio/editorial-policies/reporting-standards

[102] W. Hasselbring, L. Carr, S. Hettrick, H. Packer, and T. Tiropanis, "FAIR and open computer science research software," 2019.

[103] D. S. Katz, M. Gruenpeter, T. Honeyman, L. Hwang, M. D. Wilkinson, V. Sochat, H. Anzt, C. Goble, and f. F. S. 1, "A fresh look at FAIR for research software," *arXiv: 2101.10883 [cs.SE]*, 2021.

[104] M. Gruenpeter, S. Granger, A. Monteil, N. C. Hong, E. Breitmoser, M. Antonioletti, D. Garijo, E. G. Guardia, A. G. Beltran, C. Goble, S. Soiland-Reyes, N. Juty, and G. Mejias, "D4.4 - Guidelines for recommended metadata standard for research software within EOSC," Zenodo, Tech. Rep., Mar. 2024. [Online]. Available: https://research.manchester.ac.uk/en/publications/d44-guidelines-for-recommended-metadata-standard-for-research-sof

[105] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, May 2009. [Online]. Available: https://doi.org/10.1109/secse.2009.5069155

[106] S. Sufi, E. Bell, and A.-M. Sichani, "Report on the AHRC Digital/Software Requirements Survey," Software Sustainability Institute, Tech. Rep., Feb. 2023. [Online]. Available: https://www.software.ac.uk/blog/report-ahrc-digitalsoftware-requirements-survey

[107] D. F. Kelly, "A Software Chasm: Software Engineering and Scientific Computing," *IEEE Software*, vol. 24, no. 6, pp. 120–119, 2007. [Online]. Available: https://doi.org/10.1109/ms.2007.155

[108] T. Storer, "Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming," *ACM Computing Surveys*, vol. 50, no. 4, pp. 47:1–47:32, Aug. 2017. [Online]. Available: http://doi.acm.org/10.1145/3084225

[109] A. Johanson and W. Hasselbring, "Software Engineering for Computational Science: Past, Present, Future," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, Mar. 2018.

[110] V. C. Storey and R. L. Baskerville, "Digitalization of the natural sciences: Design science research and computational science," *Decision Support Systems*, vol. 189, p. 114368, Feb. 2025.

[111] G. V. Wilson and T. Dunne, "Macroscope: Where's the Real Bottleneck in Scientific Computing?" *American Scientist*, vol. 94, no. 1, pp. 5–6, 2006. [Online]. Available: http://ezproxy-prd.bodleian.ox.ac.uk:2154/stable/27858697

[112] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best Practices for Scientific Computing," *PLoS Biology*, vol. 12, no. 1, p. e1001745, 2014. [Online]. Available: https://doi.org/10.1371/journal.pbio.1001745

[113] V. Stodden and S. Miguez, "Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research," *Journal of Open Research Software*, vol. 2, no. 1, p. nil, 2014. [Online]. Available: https://doi.org/10.5334/jors.ay

[114] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good Enough Practices in Scientific Computing," *PLoS Computational Biology*, vol. 13, no. 6, Jun. 2017.

[115] G. Lee, S. Bacon, I. Bush, L. Fortunato, D. Gavaghan, T. Lestang, C. Morton, M. Robinson, P. Rocca-Serra, S.-A. Sansone, and H. Webb, "Barely Sufficient Practices in Scientific Computing," *Cell Patterns*, vol. 2, no. 2, Feb. 2021.

[116] T. T. W. Community, B. Arnold, L. Bowler, S. Gibson, P. Herterich, R. Higman, A. Krystalli, A. Morley, M. O'Reilly, and K. Whitaker, "The turing way: A handbook for reproducible data science," Zenodo, Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3233986

[117] G. Lee, D. Mulholland, A. Clarke, B. Mummery, and M. Hagdorn, "Is testing overkill for most research software? How do we make it easier to test scripts?" Sep. 2021. [Online]. Available: https://youtu.be/9084fOirQYo?t=928

[118] J. Carver, I. D. Cottam, C. C. Venters, R. A. Pepper, and M. Woodbridge, "Industrial Software Development Practices: Do They Work in Academia? Should RSEs Learn and Use Them?" Sep. 2021. [Online]. Available: https://youtu.be/6iKPUtpt0Rw

[119] Open Source Initiative, "The Open Source Definition," https://opensource.org/osd, Jul. 2006.

[120] M. Borg, P. Chatzipetrou, K. Wnuk, E. Alégroth, T. Gorschek, E. Papatheocharous, S. M. A. Shah, and J. Axelsson, "Selecting component sourcing options: A survey of software engineering's broader make-or-buy decisions," *Information and Software Technology*, vol. 112, pp. 18–34, Aug. 2019.

[121] N. Chue Hong, "FAIR Software? How can we make it easier to find, access, deposit and reuse software?" 2017.

[122] L. Fortunato and M. Galassi, "The case for free and open source software in research and scholarship," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2197, pp. rsta.2020.0079, 20 200 079, May 2021.

[123] B. D. Lee, "Ten Simple Rules for Documenting Scientific Software," *PLOS Computational Biology*, vol. 14, no. 12, p. e1006561, 2018. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1006561

[124] M. D. Hanwell, A. Perera, W. Turner, P. O'Leary, K. Osterdahl, B. Hoffman, and W. Schroeder, "Sustainable Software Ecosystems for Open Science," *arXiv:1309.2966 [cs]*, Sep. 2013.

[125] D. Muna, M. Alexander, A. Allen, R. Ashley, D. Asmus, R. Azzollini, M. Bannister, R. Beaton, A. Benson, G. B. Berriman, M. Bilicki, P. Boyce, J. Bridge, J. Cami, E. Cangi, X. Chen, N. Christiny, C. Clark, M. Collins, J. Comparat, N. Cook, D. Croton, I. D. Davids, É. Depagne, J. Donor, L. A. dos Santos, S. Douglas, A. Du, M. Durbin, D. Erb, D. Faes, J. G. Fernández-Trincado, A. Foley, S. Fotopoulou, S. Frimann, P. Frinchaboy, R. Garcia-Dias, A. Gawryszczak, E. George, S. Gonzalez, K. Gordon, N. Gorgone, C. Gosmeyer, K. Grasha, P. Greenfield, R. Grellmann, J. Guillochon, M. Gurwell, M. Haas, A. Hagen, D. Haggard, T. Haines, P. Hall, W. Hellwing, E. C. Herenz, S. Hinton, R. Hlozek, J. Hoffman, D. Holman, B. W. Holwerda, A. Horton, C. Hummels, D. Jacobs, J. J. Jensen, D. Jones, A. Karick, L. Kelley, M. Kenworthy, B. Kitchener, D. Klaes, S. Kohn, P. Konorski, C. Krawczyk, K. Kuehn, T. Kuutma, M. T. Lam, R. Lane, J. Liske, D. Lopez-Camara, K. Mack, S. Mangham, Q. Mao, D. J. E. Marsh, C. Mateu, L. Maurin, J. McCormac, I. Momcheva, H. Monteiro, M. Mueller, R. Munoz, R. Naidu, N. Nelson, C. Nitschelm, C. North, J. Nunez-Iglesias, S. Ogaz, R. Owen, J. Parejko, V. Patrício, J. Pepper, M. Perrin, T. Pickering, J. Piscionere, R. Pogge, R. Poleski, A. Pourtsidou, A. M. Price-Whelan, M. L. Rawls,

S. Read, G. Rees, H. Rein, T. Rice, S. Riemer-Sørensen, N. Rusomarov, S. F. Sanchez, M. Santander-García, G. Sarid, W. Schoenell, A. Scholz, R. L. Schuhmann, W. Schuster, P. Scicluna, M. Seidel, L. Shao, P. Sharma, A. Shulevski, D. Shupe, C. Sifón, B. Simmons, M. Sinha, I. Skillen, B. Soergel, T. Spriggs, S. Srinivasan, A. Stevens, O. Streicher, E. Suchyta, J. Tan, O. G. Telford, R. Thomas, C. Tonini, G. Tremblay, S. Tuttle, T. Urrutia, S. Vaughan, M. Verdugo, A. Wagner, J. Walawender, A. Wetzel, K. Willett, P. K. G. Williams, G. Yang, G. Zhu, and A. Zonca, "The Astropy Problem," 2016.

[126] I. Wiese, I. Polato, and G. Pinto, "Naming the Pain in Developing Scientific Software," *IEEE Software*, vol. 37, no. 4, pp. 75–82, 2020.

[127] M. T. Sletholt, J. Hannay, D. Pfahl, H. C. Benestad, and H. P. Langtangen, "A literature review of agile practices and their effects in scientific software development," in *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering*. ACM, May 2011, pp. 1–9. [Online]. Available: http://ezproxy-prd.bodleian.ox.ac.uk:2769/citation.cfm?id=1985782.1985784

[128] K. Beck, "A Laboratory for Teaching Object-Oriented Thinking," in *OOPSLA '89*, 1989-10-01/1989-10-06.

[129] ISACA, "CMMI Model Quick Reference Guide," ISACA, Tech. Rep., 2024.

[130] J. Segal, "The nature of evidence in empirical software engineering," in *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, 2003, pp. 40–47.

[131] D. S. Katz, K. McHenry, C. Reinking, and R. Haines, "Research software development & management in universities: Case studies from manchester's RSDS group, illinois' NCSA, and notre dame's CRC," *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*, May 2019. [Online]. Available: http://dx.doi.org/10.1109/SE4Science.2019.00009

[132] J. Segal, "When Software Engineers Met Research Scientists: A Case Study," *Empirical Software Engineering*, vol. 10, no. 4, pp. 517–536, 2005. [Online]. Available: https://doi.org/10.1007/s10664-005-3865-y

[133] R. Hoda, "Socio-Technical Grounded Theory for Software Engineering," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 3808–3832, Oct. 2022. [Online]. Available: http://arxiv.org/abs/2103.14235

[134] Efrosyni-Maria Skordaki and S. Bainbridge, "Blended Training on Scientific Software: A Study on How Scientific Data are Generated," *International Review of Research in Open and Distributed Learning*, vol. 19, no. 2, Apr. 2018. [Online]. Available: https://www.proquest.com/scholarly-journals/blended-training-on-scientific-software-study-how/docview/2056769612/se-2?accountid=13042

[135] S. G. Verhulst, "Data Stewardship Reimagined—Capacities and Competencies," Oct. 2021. [Online]. Available: https://medium.com/data-stewards-network/data-stewardship-reimagined-capacities-and-competencies-d37a0ebaf0ee

[136] European Commission. Directorate General for Research and Innovation. and PwC EU Services., "Cost-benefit analysis for FAIR research data: Cost of not having FAIR research data." Publications Office, LU, Tech. Rep., 2018. [Online]. Available: https://data.europa.eu/doi/10.2777/02999

[137] ACM Code 2018 Task Force, "ACM Code of Ethics and Professional Conduct," Jun. 2018. [Online]. Available: https://www.acm.org/code-of-ethics

[138] J. B. Tucker and R. A. Zilinskas, "The Promise and Perils of Synthetic Biology," *The New Atlantis*, no. 12, pp. 25–45, 2006.

[139] H. Åm, "'Don't make nanotechnology sexy, ensure its benefits, and be neutral': Studying the logics of new intermediary institutions in ambiguous governance contexts," *Science and Public Policy*, vol. 40, no. 4, pp. 466–478, Aug. 2013.

[140] J. Stilgoe, M. Watson, and K. Kuo, "Public Engagement with Biotechnologies Offers Lessons for the Governance of Geoengineering Research and Beyond," *PLOS Biology*, vol. 11, no. 11, p. e1001707, Nov. 2013.

[141] J. Stilgoe, R. Owen, and P. Macnaghten, "Developing a framework for responsible innovation," in *The Ethics of Nanotechnology, Geoengineering, and Clean Energy*. Routledge, 2017, pp. 347–359.

[142] G. Eden, M. Jirotka, and B. Stahl, "Responsible research and innovation: Critical reflection into the potential social consequences of ICT," in *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, May 2013, pp. 1–12.

[143] F. Ananasso, S. Farruggia, R. Provedel, and M. Sebillo, "Responsible Research and Innovation in Open Health and Open Science. Open Science: Open and Toll-Free Data Age Open Health: From P2P (Patient to Physician) to P2E (Person to Eco-System)," in *Governance and Sustainability of Responsible Research and Innovation Processes: Cases and Experiences*, ser. SpringerBriefs in Research and Innovation Governance. Springer International Publishing, 2018, pp. 27–34.

[144] ——, "Responsible Research and Open Innovation in Geospatial Applications: Some Good Practices for Smart Communities," in *Governance and Sustainability of Responsible Research and Innovation Processes*, ser. SpringerBriefs in Research and Innovation Governance. Springer International Publishing, 2018.

[145] Engineering and Physical Sciences Research Council, "Anticipate, reflect, engage, and act (AREA)." [Online]. Available: https://epsrc.ukri.org/research/framework/area/

[146] European Commission and Directorate-General for Research and Innovation, *Responsible research and innovation – Europe's ability to respond to societal challenges*. Publications Office, 2014.

[147] N. C. Borle, M. Feghhi, E. Stroulia, R. Greiner, and A. Hindle, "Analyzing the effects of test driven development in GitHub," *Empirical Software Engineering*, vol. 23, no. 4, pp. 1931–1958, Aug. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9576-3

[148] G. Wilson, "Analyzing the effects of test driven development in GitHub," Sep. 2021. [Online]. Available: https://neverworkintheory.org/2021/09/16/analyzing-the-effects-of-tdd-in-github.html

[149] D. Fucci, G. Scanniello, S. Romano, M. Shepperd, B. Sigweni, F. Uyaguari, B. Turhan, N. Juristo, and M. Oivo, "An external replication on the effects of test-driven development using a multi-site blind analysis approach," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2961111.2962592

[150] G. Wilson, "Test-Driven Development," Oct. 2016. [Online]. Available: https://neverworkintheory.org/2016/10/05/test-driven-development.html

[151] R. C. Martin, "The Jury is In," in *The Clean Coder: A Code of Conduct for Professional Programmers*. Pearson Education, Inc., 2011.

[152] J. Kruger and D. Dunning, "Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments," *Journal of Personality and Social Psychology*, vol. 77, no. 6, pp. 1121–1134, Dec. 1999.

[153] G. Pinto, I. Wiese, and L. F. Dias, "How do scientists develop scientific software? An external replication," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 582–591.

[154] G. Reggio, M. Leotta, M. Cerioli, R. Spalazzese, and F. Alkhabbas, "What are IoT systems for real? An experts' survey on software engineering aspects," *Internet of Things*, vol. 12, p. 100313, Dec. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S254266052030144X

[155] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Chapel Hill, North Carolina, U.S.A.: Addison-Wesley, 1975.

[156] G. M. Weinberg, *The Psychology of Computer Programming*. Van Nostrand Reinhold Company, Feb. 1972.

[157] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, 2nd ed. Dorset House, 1999.

[158] M. Petre, J. Buckley, L. Church, M.-A. Storey, and T. Zimmermann, "Behavioral Science of Software Engineering," *IEEE Software*, vol. 37, no. 6, pp. 21–25, 2020.

[159] M.-A. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, "The Who, What, How of Software Engineering Research: A Socio-Technical Framework," 2020.

[160] Y. Dittrich, "What does it mean to use a method? Towards a practice theory for software engineering," *Information and Software Technology*, vol. 70, pp. 220–231, Feb. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095058491500124X

[161] B. Fitzgerald, "An empirical investigation into the adoption of systems development methodologies," *Information & Management*, vol. 34, no. 6, pp. 317–328, Dec. 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037872069800072X

[162] B. G. Glaser and A. L. Strauss, *Discovery of Grounded Theory: Strategies for Qualitative Research*. Somerset, UNITED STATES: Taylor & Francis Group, 1999. [Online]. Available: http://ebookcentral.proquest.com/lib/oxford/detail.action?docID=4905885

[163] K. L. Rieger, "Discriminating among grounded theory approaches," *Nursing Inquiry*, vol. 26, no. 1, p. e12261, Jan. 2019. [Online]. Available: https://doi.org/10.1111/nin.12261

[164] M. Birks and J. Mills, *Grounded Theory: A Practical Guide*, 2nd ed. SAGE Publications Ltd, 2015.

[165] K. Charmaz, *Constructing Grounded Theory : A Practical Guide through Qualitative Analysis*, ser. Introducing Qualitative Methods. London: Sage, 2006.

[166] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 120–131. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2102/10.1145/2884781.2884833

[167] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing Agile teams," *Empirical Software Engineering*, vol. 17, no. 6, pp. 609–639, Dec. 2012. [Online]. Available: https://doi.org/10.1007/s10664-011-9161-0

[168] D. Wells, "The Customer is Always Available," 1999. [Online]. Available: http://www.extremeprogramming.org/rules/customer.html

[169] A. Martin, R. Biddle, and J. Noble, "The XP customer team: A grounded theory," in *2009 Agile Conference*, 2009, pp. 57–64.

[170] I. Cavar, ""The Good, the Bad, and the Ugly" of Professions: Overview of the Theoretical Developments in the Sociology of Professions," *Interdisciplinary Description of Complex Systems*, vol. 19, no. 1, pp. 80–93, 2021. [Online]. Available: http://indecs.eu/index.php?s=x&y=2021&p=80-93

[171] E. H. Gorman and R. L. Sandefur, ""Golden Age," Quiescence, and Revival: How the Sociology of Professions Became the Study of Knowledge-Based Work," *Work and Occupations*, vol. 38, no. 3, pp. 275–302, Aug. 2011. [Online]. Available: https://doi.org/10.1177/0730888411417565

[172] W. F. Atchison, E. J. Schweppe, W. Viavant, D. M. Young, S. D. Conte, J. W. Hamblen, T. E. Hull, T. A. Keenan, W. B. Kehl, E. J. McCluskey, S. O. Navarro, and W. C. Rheinboldt, "Curriculum 68: Recommendations for academic programs in computer science: A report of the ACM curriculum committee on computer science," *Communications of the ACM*, vol. 11, no. 3, pp. 151–197, Mar. 1968. [Online]. Available: http://portal.acm.org/citation.cfm?doid=362929.362976

[173] R. Dingwall, "'In the Beginning was the Work' Reflections on the Genesis of Occupations," *The Sociological Review*, vol. 31, no. 4, pp. 605–624, Nov. 1983. [Online]. Available: http://journals.sagepub.com/doi/10.1111/j.1467-954X.1983.tb00723.x

[174] N. Ensmenger, "The 'question of professionalism' in the computer fields," *IEEE Annals of the History of Computing*, vol. 23, no. 4, pp. 56–74, 2001.

[175] M. Franco-Santos, M. Nalick, P. Rivera-Torres, and L. Gomez-Mejia, "Governance and Well-being in Academia: Negative Consequences of Applying an Agency Theory Logic in Higher Education." *British Journal of Management*, vol. 28, no. 4, pp. 711–730, Oct. 2017. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=126171629&site=ehost-live&authtype=ip,uid

[176] C. Whitchurch, "Shifting Identities and Blurring Boundaries: The Emergence of Third Space Professionals in UK Higher Education," *Higher Education Quarterly*, vol. 62, no. 4, pp. 377–396, Oct. 2008. [Online]. Available: https://onlinelibrary-wiley-com.ezproxy-prd.bodleian.ox.ac.uk/doi/full/10.1111/j.1468-2273.2008.00387.x

[177] L. A. Perlow, "The Time Famine: Toward a Sociology of Work Time," *Administrative Science Quarterly*, vol. 44, no. 1, p. 57, Mar. 1999. [Online]. Available: https://www.proquest.com/scholarly-journals/time-famine-toward-sociology-work/docview/1554070820/se-2?accountid=13042

[178] K. Sang, A. Powell, R. Finkel, and J. Richards, "'Being an academic is not a 9–5 job': Long working hours and the 'ideal worker' in UK academia," *Labour & Industry: a journal of the social and economic relations of work*, vol. 25, no. 3, pp. 235–249, Jul. 2015. [Online]. Available: https://doi.org/10.1080/10301763.2015.1081723

[179] N. Kameo, "A Culture of Uncertainty: Interaction and Organizational Memory in Software Engineering Teams under a Productivity Scheme," *Organization Studies*, vol. 38, no. 6, pp. 733–752, Jun. 2017. [Online]. Available: https://doi.org/10.1177/0170840616685357

[180] R. Schreiber and M. MacDonald, "Keeping Vigil over the Patient: A grounded theory of nurse anaesthesia practice," *Journal of Advanced Nursing*, vol. 66, no. 3, pp. 552–561, Mar. 2010. [Online]. Available: https://doi.org/10.1111/j.1365-2648.2009.05207.x

[181] R. S. Schreiber and M. A. MacDonald, "Keeping vigil over the profession: A grounded theory of the context of nurse anaesthesia practice," *BMC Nursing*, vol. 9, no. 1, p. 13, Jul. 2010. [Online]. Available: https://doi.org/10.1186/1472-6955-9-13

[182] S. Calvert, E. Smythe, and B. McKenzie-Green, ""Working towards being ready": A grounded theory study of how practising midwives maintain their ongoing competence to practise their profession," *Midwifery*, vol. 50, pp. 9–15, Jul. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0266613817301924

[183] M. Alvesson and A. Spicer, "Neo-Institutional Theory and Organization Studies: A Mid-Life Crisis?" *Organization*

Studies, vol. 40, no. 2, pp. 199–218, Feb. 2019. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=bsu&AN=134612410&site=ehost-live&authtype=ip,uid

[184] R. Suddaby, "Challenges for Institutional Theory," 2010. [Online]. Available: https://journals-sagepub-com.ezproxy-prd.bodleian.ox.ac.uk/doi/abs/10.1177/1056492609347564

[185] J. W. Meyer and B. Rowan, "Chapter 3 Institutionalized Organizations: Formal Structure as Myth and Ceremony," in *The New Economic Sociology*, F. Dobbin, Ed. Princeton University Press, Dec. 2004, pp. 86–110. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/9780691229270-004/html

[186] R. Suddaby, D. Seidl, and J. K. Lê, "Strategy-as-practice meets neo-institutional theory," 2013. [Online]. Available: https://journals-sagepub-com.ezproxy-prd.bodleian.ox.ac.uk/doi/epub/10.1177/1476127013497618

[187] M. C. Suchman, "Managing legitimacy: Strategic and institutional approaches," *Academy of Management. The Academy of Management Review*, vol. 20, no. 3, p. 571, Jul. 1995. [Online]. Available: https://www.proquest.com/scholarly-journals/managing-legitimacy-strategic-institutional/docview/210941848/se-2?accountid=13042

[188] P. J. DiMaggio and W. W. Powell, "The Iron Cage Revisited: Institutional Isomorphism and Collective Rationality in Organizational Fields," *American Sociological Review*, vol. 48, no. 2, pp. 147–160, 1983. [Online]. Available: https://www.jstor.org/stable/2095101

[189] M. Saks, "A review of theories of professions, organizations and society: The case for neo-Weberianism, neo-institutionalism and eclecticism," *Journal of Professions and Organization*, vol. 3, no. 2, pp. 170–187, Sep. 2016. [Online]. Available: https://doi.org/10.1093/jpo/jow005

[190] E. Wenger, *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press, Jul. 1998. [Online]. Available: https://www.cambridge.org/highereducation/books/communities-of-practice/724C22A03B12D11DFC345EEF0AD3F22A

[191] ——, "Knowledge Management as a donut," *Ivey Business Journal*, no. January/February 2004, p. 14, Jan. 2004.

[192] E. Wenger, B. Trayner, and M. de Laat, "Promoting and assessing value creation in communities and networks: A conceptual framework," Ruud de Moor Centrum, Netherlands Open University, Tech. Rep., 2011.

[193] E. Orhun and J. Hopple, "Theoretical frameworks for knowledge sharing in a community of practice," in *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, ser. EATIS '08. New York, NY, USA: Association for Computing Machinery, Sep. 2008, pp. 1–7. [Online]. Available: https://doi.org/10.1145/1621087.1621091

[194] J. Nagy and T. Burch, "Communities of Practice in Academe (CoP-iA): Understanding academic work practices to enable knowledge building capacities in corporate universities," *Oxford Review of Education*, vol. 35, no. 2, pp. 227–247, 2009. [Online]. Available: https://www.jstor.org/stable/27784556

[195] C. Y. Shim, Y. Kim, and I. Kim, "Designing Framework for Evaluating Community of Practice for Software Engineering Project Management," in *2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*, Dec. 2023, pp. 575–578. [Online]. Available: https://ieeexplore.ieee.org/document/10471839/references#references

[196] S. Kumar and C. Wallace, "Patterns of Identity and Interaction in an Agile Community of Practice," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2019. [Online]. Available: https://ieeexplore-ieee-org.ezproxy-prd.bodleian.ox.ac.uk/document/8816883

[197] V. Garousi, M. Borg, and M. Oivo, "Practical relevance of software engineering research: Synthesizing the community's voice," *Empirical Software Engineering*, vol. 25, no. 3, pp. 1687–1754, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1007/s10664-020-09803-0

[198] J. W. Creswell and V. L. Plano Clark, "Core Mixed Methods Designs," in *Designing and Conducting Mixed Methods Research*, 3rd ed. SAGE Publications Ltd, Sep. 2017, pp. 51–98.

[199] R. M. de Mello and G. H. Travassos, "Surveys in software engineering: Identifying representative samples," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2102/10.1145/2961111.2962632

[200] J. S. Molléri, K. Petersen, and E. Mendes, "Survey guidelines in software engineering: An annotated review," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://ezproxy-prd.bodleian.ox.ac.uk:2102/10.1145/2961111.2962619

[201] M. Kasunic, "Designing an Effective Survey," Carnegie Mellon University, Report, Sep. 2005.

[202] J. Linåker, S. M. Sulaman, R. Maiani de Mello, and M. Höst, "Guidelines for Conducting Surveys in Software Engineering," Lund University, Tech. Rep., 2015.

[203] S. Sudman and N. M. Bradburn, *Asking Questions : A Practical Guide to Questionnaire Design*, ser. Jossey-Bass Social and Behavioral Science Series. San Francisco ; London: Jossey-Bass, 1982.

[204] University of Oxford, "Facts and figures | University of Oxford," https://www.ox.ac.uk/about/facts-and-figures, 2025.

[205] University of Oxford - SABS R^3 CDT, "Students," https://www.sabsr3.ox.ac.uk/students, 2024.

[206] Chatham House, "Chatham House Rule," 2002. [Online]. Available: https://www.chathamhouse.org/about-us/chatham-house-rule

[207] A. K. Goodboy, M. M. Martin, C. B. Mills, and C. V. Clark-Gordon, "Workplace Bullying in Academia: A Conditional Process Model," *Management Communication Quarterly*, vol. 36, no. 4, pp. 664–687, Nov. 2022. [Online]. Available: http://journals.sagepub.com/doi/10.1177/08933189221103625

[208] Advanced Research Computing Centre, "How to publish research software." [Online]. Available: https://www.bristol.ac.uk/acrc/research-software-engineering/software-howtos/how-to-publish-software/

[209] P. Ayers, "Citing & publishing software:publishing research software." [Online]. Available: https://libguides.mit.edu/software

[210] S. Stall, G. Bilder, M. Cannon, N. Chue Hong, S. Edmunds, C. C. Erdmann, M. Evans, R. Farmer, P. Feeney, M. Friedman, M. Giampoala, R. B. Hanson, M. Harrison, D. Karaiskos, D. S. Katz, V. Letizia, V. Lizzi, C. MacCallum, A. Muench, K. Perry, H. Ratner, U. Schindler, B. Sedora, M. Stockhause, R. Townsend, J. Yeston, and T. Clark, "Journal Production Guidance for Software and Data Citations," *Scientific Data*, vol. 10, no. 1, p. 656, Sep. 2023. [Online]. Available: https://www.nature.com/articles/s41597-023-02491-7

[211] Education in England, "Dearing Report: Higher Education in the Learning Society," 1997. [Online]. Available: https://education-uk.org/documents/dearing1997/dearing1997.html

[212] T. Savage, "Creative arts technicians in academia: To transition or not to transition?" *Art, Design & Communication in Higher Education*, vol. 17, no. 2, pp. 237–253, Oct. 2018. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=aft&AN=132316459&site=ehost-live&authtype=ip,uid

[213] L. C. Kurtz, S. Trainer, M. Beresford, A. Wutich, and A. Brewis, "Blogs as Elusive Ethnographic Texts: Methodological and Ethical Challenges in Qualitative Online Research," *International Journal of Qualitative Methods*, vol. 16, no. 1, p. 160940691770579, Dec. 2017. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1609406917705796

[214] T. Lawrence, R. Suddaby, and B. Leca, "Institutional Work: Refocusing Institutional Studies of Organization," *Journal of Management Inquiry*, vol. 20, no. 1, pp. 52–58, Mar. 2011. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1056492610387222

[215] W. W. Powell and J. A. Colyvas, *Microfoundations of Institutional Theory*. 1 Oliver's Yard, 55 City Road, London EC1Y 1SP United Kingdom: SAGE Publications Ltd, 2008, pp. 276–298. [Online]. Available: https://sk.sagepub.com/reference/hdbk_orginstitution/n11.xml

[216] P. Bourque, R. E. Fairley, and IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society Press, 2014.

[217] Software Carpentry, "Our Lessons," Dec. 2023. [Online]. Available: https://software-carpentry.org//lessons/

[218] Openscapes, "Resources — Openscapes," 2023. [Online]. Available: https://openscapes.org/resources

[219] M. T. Schönborn, "Adopting Software Engineering Concepts in Scientific Research: Insights from Physicists and Mathematicians Turned Consultants," *Computing in Science & Engineering*, vol. 25, no. 4, pp. 25–33, Jul. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10309169?source=tocalert&dld=aWFtbGVlZy5jb20%3D

[220] W. Cunningham, "The WyCash portfolio management system," in *OOPSLA '92*, Vancouver, British Columbia, Canada, 1992-10-05/1992-10-10, pp. 29–30.

[221] R. G. Carmel and M. W. Paul, "Mentoring and coaching in academia: Reflections on a mentoring/coaching relationship," *Policy Futures in Education*, vol. 13, no. 4, pp. 479–491, Jun. 2015. [Online]. Available: https://doi.org/10.1177/1478210315578562

[222] L. J. Holt and M. J. Lopez, "Characteristics and Correlates of Supportive Peer Mentoring: A Mixed Methods Study," *Mentoring & Tutoring: Partnership in Learning*, vol. 22, no. 5, pp. 415–432, Oct. 2014. [Online]. Available: https://doi.org/10.1080/13611267.2014.983326

[223] C. Goble, "What is Research Software? And why is it critical to the research endeavour?" in *2nd International Funders Workshop: The Future of Research Software*, 2023. [Online]. Available: https://zenodo.org/records/10138709

[224] F. Psomopoulos, S. Capella-Gutierrez, L. Portell-Silva, and N. Pechlivanis, "EOSC-EVERSE: Paving the way towards a European Virtual Institute for Research Software Excellence," Jan. 2024.

[225] K. Wiegers and C. Hokanson, *Software Requirements Essentials: Core Practices for Successful Business Analysis*. Pearson Education, Inc., 2023.

[226] M. Felderer, M. Goedicke, L. Grunske, W. Hasselbring, A.-L. Lamprecht, and B. Rumpe, "Investigating Research Software Engineering: Toward RSE Research," *Commun. ACM*, vol. 68, no. 2, pp. 20–23, Jan. 2025.

[227] M. A. Heroux, "Research Software Science: Expanding the Impact of Research Software Engineering," *Computing in Science & Engineering*, vol. 24, no. 6, pp. 22–27, Nov. 2022.

[228] J. Corral-García, C. Gómez-Martín, J.-L. González-Sánchez, and D. Cortés-Polo, "Development of Scientific Applications with High-Performance Computing through a Component-Based and Aspect-Oriented Methodology," *International Journal of Advanced Computer Science*, vol. 3, no. 8, pp. 400–408, Jul. 2013.

[229] M. C. Feathers, *Working Effectively with Legacy Code*, ser. Robert C. Martin Series. Prentice Hall, 2005.

[230] R. C. Jiménez, M. Kuzak, M. Alhamdoosh, M. Barker, B. Batut, M. Borg, S. Capella-Gutierrez, N. C. Hong, M. Cook, M. Corpas, M. Flannery, L. Garcia, J. L. Gelpí, S. Gladman, C. Goble, M. G. Ferreiro, A. Gonzalez-Beltran, P. C. Griffin, B. Grüning, J. Hagberg, P. Holub, R. Hooft, J. Ison, D. S. Katz, B. Leskošek, F. L. Gómez, L. J. Oliveira, D. Mellor, R. Mosbergen, N. Mulder, Y. Perez-Riverol, R. Pergl, H. Pichler, B. Pope, F. Sanz, M. V. Schneider, V. Stodden, R. Suchecki, R. S. Vařeková, H.-A. Talvik, I. Todorov, A. Treloar, S. Tyagi, M. van Gompel, D. Vaughan, A. Via, X. Wang, N. S. Watson-Haigh, and S. Crouch, "Four Simple Recommendations To Encourage Best Practices in Research Software," *F1000Research*, vol. 6, no. nil, p. 876, 2017.

[231] J. Cohen, D. S. Katz, M. Barker, N. Chue Hong, R. Haines, and C. Jay, "The four pillars of research software engineering," *IEEE Software*, vol. 38, no. 1, pp. 97–105, 2021.

[232] The CRediT Project, "CRediT," https://credit.niso.org/, 2025.

[233] C. Jay, R. Haines, and D. S. Katz, "Software Must be Recognised as an Important Output of Scholarly Research," *International Journal of Digital Curation*, vol. 16, no. 1, p. 6, Dec. 2021.

[234] The Hidden REF Project, "The Hidden REF," https://hidden-ref.org/, 2025.

[235] V. I. Lenin, "What is to be Done?: Burning Questions of our Movement," Mar. 1902. [Online]. Available: https://www.marxists.org/archive/lenin/works/1901/witbd/