Report from Dagstuhl Perspectives Workshop 16252

Engineering Academic Software

Edited by

Carole Goble[1], James Howison[2], Claude Kirchner[3], Oscar Nierstrasz[4], and Jurgen J. Vinju[5]

1   **The University of Manchester, UK,**
    `http://www.manchester.ac.uk/research/Carole.goble/`
2   **The University of Texas at Austin, USA,**
    `https://www.ischool.utexas.edu/people/person_details?PersonID=175`
3   **Inria – Le Chesnay, FR,** `http://www.loria.fr/~ckirchne/`
4   **University of Bern, CH,** `http://scg.unibe.ch/staff/oscar`
5   **Centrum Wiskunde & Informatica, NL,** `http://homepages.cwi.nl/~jurgenv`

──── **Abstract** ────

This report documents the program and the outcomes of Dagstuhl Perspectives Workshop 16252
"Engineering Academic Software".

1   Executive Summary

Carole Goble
James Howison
Claude Kirchner
Oscar Nierstrasz
Jurgen J. Vinju

This Dagstuhl Perspectives Workshop brought together activists, experts and stakeholders
on the subject of high quality software produced in an academic context.[1] Our current
dependence on software across the sciences is already significant, yet there are still more
opportunities to be explored and risks to be overcome. The academic context is unique
in terms of its personnel, its goals of exploring the unknown and its demands on quality
assurance and reproducibility.

We refer to the IEEE Internet Computing article "Better Software, Better Research" [1]
which motivated the topic. In this workshop we took the following perspective of a research
team which is in either or both of the following situations:

- consuming or producing software as *an output* of the academic process;
- consuming or producing software as *a component* of the research methods.

---

[1] We include any software which is part of either research processes and/or output, while excluding more
generic administrative software for research and education management.

Society is now in the tricky situation where several deeply established academic fields (*e.g.*, physics, biology, mathematics) are shifting towards dependence on software, programming technology and software engineering methodology which are backed only by young and rapidly evolving fields of research (computer science and software engineering). Full accountability and even validity of software-based research results are now duly being challenged.

With the outputs of this interactive and productive perspectives workshop, we strive to contribute in a positive manner to the above challenges. We formulated taxonomies with definitions to clarify the domain, we co-authored concrete policy and process documents to improve the status and recognition of academic software development and academic software engineers, and finally we formulated a list of 18 concrete declarations of intent ("I will" pledges). This list was presented to the WSSSPE community [2] in September 2016 to acquire feedback and it will be the backbone of the Dagstuhl Manifesto document we are editing. It serves to motivate change by proposing policy changes with concrete actions and instilling positive attitudes towards academic software.

Participants.   The participants of the workshop came from three major groups. The first group consists of active and visible members of the *global academic software engineering community*. They represent (formal) institutions such as the Software Sustainability Institute, the Software Carpentry Foundation, and eScience and data science centers from across the globe. The second group contributed researchers in *empirical software engineering*, with a specific eye on studying the principles and practices of academic software engineering. The final group contributed *researchers as an audience*: software engineering researchers with a long experience in engineering software for software itself or software for specific academic research fields.

We found that without exception the participants were strongly motivated and able to actively contribute to the proceedings of the workshop; the mix of people proved to be well-balanced. This balance is an accomplishment, given that invitees from computer science were far more likely to know of Dagstuhl workshops than other groups. To attest to our outcomes we've selectively listed three (paraphrased) verbal statements here:

- "The workshop was a transformational experience for me; I've learned an entire new perspective on my field and I intend to apply the insights in my daily practice."
- "I had an epiphany yesterday after dinner; now I understand how to connect the data science research at my university to the computer science department."
- "Before the workshop I had no idea so many initiatives were already underway in [improving] academic software engineering; this has given my understanding of the challenges a real boost and I know what the some of the next steps to take are."

Schedule.   The schedule of the workshop was designed to maximize both interactive discussion and work towards tangible outputs. Key points were: to start the day with inspiring presentations to set the stage, then to have at least 40% of the day time allocated to free discussion time, and to explicitly share successes (output) of each day's breakout groups in a plenary session.

The workshop started on Monday with a quick and tightly timed round of 2 minute personal introductions. Otherwise on Monday, Tuesday and Thursday the program was structured equally: in the morning we would have plenary presentations which included exploratory discussions. These sessions were meant to bring everybody up-to-speed with ongoing and past initiatives. During and after lunch we used a board with sticky notes to define break-out groups. Each break-out group was centred around a specific discussion topic and (usually) a specific idea for an output document was associated with it. After

coffee we would go back to the same break-out group to collaboratively record the notes and lessons from each group (stored in a shared online document). Between 17:00 and 18:00 we reconvened and harvested the results of each breakout group with the others. People could and did freely switch between breakout groups but this was not a common thing.

On Wednesday we had an "open-mic" session with 8 presentations of around 10 minutes, sharing experiences and results, before we had a long walk in the surroundings. The organizers also designed an initial skeleton structure and ideas for the manifesto that day.

On Thursday afternoon and Friday morning we all worked together on our Dagstuhl Manifesto by first reworking our notes into the ideas around the manifesto, specifically a list of "I will" pledges with references and motivation. Finally, Friday afternoon a small remaining group re-ordered the group's manifesto notes into a well-structured list of 18 pledges. Two of the organizers remained to continue to edit the current report and the manifesto document.

Output. Output documents of the workshop are organized under the "DagstuhlEAS" organisation on GitHub.[2] This currently features 6 draft documents, including the current report and (a) the manifesto, (b) the Research Software Engineering Handbook, (c) a Literature Survey, (d) a Taxonomy on Software Credit Roles, and (e) a Software Award Proposal. Next to these documents, an R&D project proposal was produced on measuring the impact of academic software.

The remainder of this document summarizes the morning sessions by listing the abstracts of each talk, the afternoon breakouts by describing each topic and its results, and finally the research questions on the topic of engineering academic software we have collected.

References

1 Carole Goble. Better software, better research. *IEEE Internet Computing*, 18(5):4–8, Sep 2014.
2 Alice Allen, Cecilia Aragon, Christophe Becker, Jeffrey C. Carver, Andrei Chis, Benoit Combemale, Mike Croucher, Kevin Crowston, Daniel Garijo, Ashish Gehani, Carole Goble, Robert Haines, Robert Hirschfeld, James Howison, Kathryn Huff, Caroline Jay, Daniel S. Katz, Claude Kirchner, Kateryna Kuksenok, Ralf Lämmel, Oscar Nierstrasz, Matthew Turk, Rob van Nieuwpoort, Matthew Vaughn, and Jurgen Vinju. Lightning talk: "I solemnly pledge" – a manifesto for personal responsibility in the engineering of academic software. In *Proceedings of the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4)*.

---

[2] https://github.com/DagstuhlEAS

2   Table of Contents

3   Overview of Talks

These are the talks presented in the morning sessions of the workshop, chronologically ordered.

3.1   Sustainable Software for Science

Daniel S. Katz (University of Illinois at Urbana-Champaign, USA)

Progress in scientific research depends on the quality and accessibility of research software at all levels. It is now critical to address many new challenges related to the development, deployment, maintenance, and sustainability of open-use research software: the software upon which specific research results rely. *Open-use software* means that the software is *widely accessible* (whether open source, shareware, or commercial). *Research software* means that the choice of software is *essential to specific research results*; using different software could produce different results.

In addition, it is essential that scientists, researchers, and students are able to learn and adopt a new set of software-related skills and methodologies. Established researchers are already acquiring some of these skills, and in particular, a specialized class of software developers is emerging in academic environments who are an integral and embedded part of successful research teams. WSSSPE[3] provides a forum for discussion of these challenges, including both positions and experiences, and a forum for the community to assemble and act.

This talk focused on the Third Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE3) [1]. It summarized the discussions, future steps, organization, and status of a set of self-organized working groups on topics including developing pathways to funding scientific software; constructing useful common metrics for crediting software stakeholders; identifying principles for sustainable software engineering design; reaching out to research software organizations around the world; and building communities for software sustainability. Some of these groups have executed these activities that they scheduled, some have in part, and others have not. A point of discussion was why these groups came to these points, and how the WSSSPE community can encourage groups to act.

References

1   Daniel S. Katz, Sou-Cheng T. Choi, Kyle E. Niemeyer, James Hetherington, Frank Löffler, Dan Gunter, Ray Idaszak, Steven R. Brandt, Mark A. Miller, Sandra Gesing, Nick D. Jones, Nic Weber, Suresh Marru, Gabrielle Allen, Birgit Penzenstadler, Colin C. Venters, Ethan Davis, Lorraine Hwang, Ilian Todorov, Abani Patra, and Miguel de Val-Borro. Report on the third workshop on sustainable software for science: Practice and experiences (WSSSPE3). *Journal of Open Research Software*, 4(1):e37, 2016.

---

[3] http://wssspe.researchcomputing.org.uk

3.2 Supporting Research Software Engineering

Mike Croucher (University of Sheffield, GB)

"Long Tail Science" – attributed to Jim Downing of the Unilever Centre for Molecular Informatics – refers to the large number of small research units that perform a huge amount of research. Much of this research involves the generation of code by relatively untrained and inexperienced programmers.

In this talk, Croucher described the challenges of working as a Research Software Engineer who supports these programmers using the University of Sheffield as a case study and introduced the efforts to build a community ("a union") of Research Software Engineers in the UK.[4]

3.3 Sustainability Design

Christoph Becker (University of Toronto, CA)

Sustainability – the "capacity to endure" – has emerged as a challenge with transformative impact on many disciplines and professions, including software engineering. It requires simultaneous consideration of at least five dimensions: environmental resources, social and individual well-being, economic prosperity, and long-term technical viability. This requires a cross-disciplinary approach to research and design that moves beyond narrow-minded solutionism. It emphasizes that "wicked problems" cannot easily be reduced to puzzle pieces. Systems thinking is needed as much as computational problem solving to achieve an integrated understanding of socio-technical systems. Shifts like that do not come easily, and for most systems, the hidden sustainability effects of past decisions in systems design are unknown. We can call this a system's "sustainability debt".

In this talk, Becker described how synergies across a range of disciplines united by the need for new design approaches focused on sustainability led to the Karlskrona Manifesto for Sustainability Design.[5] He characterized principles of sustainability design and the key influence of requirements activities on the sustainability debt of a system under design. He presented recent efforts to develop this area of research, including an interview study of software professionals, as a starting point to a discussion of barriers and opportunities for sustainability design research.

---

[4] http://www.rse.ac.uk/who.html
[5] http://www.sustainabilitydesign.org

3.4   What We Have Learned about Using Software Engineering Practices in Scientific Software

Jeffrey Carver (University of Alabama, USA)

The increase in the importance of Scientific Software motivates the need to identify and understand which software engineering (SE) practices are appropriate. Because of the uniqueness of the scientific software domain, existing SE tools and techniques developed for the business/IT community are often not efficient or effective. Appropriate SE solutions must account for the salient characteristics of the scientific software development environment. To identify these solutions, members of the SE community must interact with members of the scientific software community. In this presentation, Carver discussed the findings from a series of case studies of scientific software projects, an ongoing workshop series, and interactions between his research group and scientific software projects.

3.5   Lessons from the YT project

Matthew Turk (University of Illinois Urbana-Champaign, USA)

In this talk, Turk described the engineering practices, both social and technical, around the YT project.[6] He described the positive aspects and the failure modes, and how the YT team attempted to route around these failure modes.

3.6   Software as Academic Output

Caroline Jay and Robert Haines (The University of Manchester, GB)

Software is now considered to be an output of academic research in its own right: venues such as SoftwareX[7] and the Journal of Open Research Software[8] highlight it as a primary contribution, and the UK Research Council includes a software category in the ResearchFish[9] application used to collect the outcomes of research projects. This phenomenon is still fairly recent, however, and two questions arise when trying to determine the validity of – or, arguably, requirement for – software as a product of the research process: (i) when should it be considered an output, and (ii) what form should that output take?

To determine when software should be considered an output, we must consider its role in the research process. Is it a tool for supporting the work, or does it represent the research

---

[6] http://yt-project.org
[7] http://www.journals.elsevier.com/softwarex
[8] http://openresearchsoftware.metajnl.com/
[9] http://www.researchfish.com

itself? To a computer scientist in the field of workflow management, the software would be considered a direct output, integral to the research. To a biologist, this same software would be considered a tool: useful for analyzing results, but not in itself an output of the research. For a bioinformatician both using and developing the tool, the answer is somewhere in the middle: whilst the core research may be in the life science domain, the modifications made to the tool as a result of this work could also be considered an output, advancing workflow management.

If the software is integral to the research – and therefore a potential output – what form should that output take? The FAIRDOM project[10] supports computational research that is FAIR: Findable, Accessible, Interoperable, Reusable. We suggest a modified version of these principles can be usefully applied to software too: it should be Findable, Accessible, Reusable and Extensible. To be findable, software must be searchable and discoverable by others, preferably via a persistent identifier. Accessible software can be viewed and downloaded by others. Reusable software can be re-run, potentially with other input data. Finally, Extensible software can be modified or extended to deal with new situations; to achieve this, the source code should be available.

The FARE principles are a starting point for defining best practice, or the "gold standard" for academic software outputs. An exemplar of the application of these principles is described in the authors' recent paper, "ABC: Using Object Tracking to Automate Behavioural Coding" [1], published at the 2016 ACM CHI conference. The source code is openly available on GitHub, making it accessible and extensible, and both this and the software environment (in a Docker container), are identified by DOIs, making the software findable and reusable.

Following the FARE principles will help to ensure that software intended to be a primary output of research is fit for purpose. Applying them in any situation where software is developed as part of research – whether it is considered a primary output or not – is also recommended, to help ensure that the resulting research is robust and reproducible.

References

1   Aitor Apaolaza, Robert Haines, Amaia Aizpurua, Andy Brown, Michael Evans, Stephen Jolly, Simon Harper, and Caroline Jay. ABC: Using Object Tracking to Automate Behavioural Coding. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems – CHI EA'16*. Association for Computing Machinery (ACM), 2016.

3.7   Software Heritage

Claude Kirchner (Inria – Le Chesnay, FR)

In this talk Kirchner introduced the *Software Heritage* project leaded by Roberto Di Cosmo,[11] in the week just before its launch. A quote from the web site explains the concept and the goals:

"Software Heritage collects and preserves software in source code form, because

---

[10] http://fair-dom.org
[11] http://www.softwareheritage.org

software embodies our technical and scientific knowledge and humanity cannot afford the risk of losing it. Software is a precious part of our cultural heritage. We curate and make accessible all the software we collect, because only by sharing it can we guarantee its preservation in the very long term."

3.8 Software Metadata: Describing "Dark Software" in Geosciences

Daniel Garijo (Technical University of Madrid, ES)

In this talk Garijo provided an overview of the current state of the art for software description in Geosciences, along with an approach to facilitate this task in OntoSoft, a distributed semantic registry for scientific software.[12] Three key aspects of OntoSoft are: a software metadata ontology designed for scientists, a distributed approach to software registries that targets communities of interest, and metadata crowdsourcing through access control. Software metadata is organized using the OntoSoft ontology, designed to support scientists to share, document, and reuse software, and organized along six dimensions: identify software, understand and assess software, execute software, get support for the software, do research with the software, and update the software.

3.9 Organising a Research Team around the Research Software around the Research Team in Software Engineering

Jurgen J. Vinju (Centrum Wiskunde & Informatica, NL)

Vinju's talk was about the motivation, experiences and lessons learned around the SWAT research group at CWI and its core product used for research and transfer, the meta-programming language and platform, Rascal,[13] which is hosted from the open-source organisation Use The Source.[14]

3.10 Software Citation – Principles, Discussion, and Metadata

Daniel S. Katz (University of Illinois at Urbana-Champaign, USA)

Katz presented an overview of work done by the Force11 Software Citation Working Group[15] [1]. This work includes rationales for citing software, information on the WSSSPE

---

[12] http://www.ontosoft.org/portal/
[13] http://www.rascal-mpl.org
[14] http://www.usethesource.io
[15] https://www.force11.org/group/software-citation-working-group

and Force11 groups involved in developing software citation principles, and the process used to develop them. It also includes the six principles themselves:

1. the importance of software,
2. the need to credit and attribute the contributions software makes to research,
3. to be able to uniquely identify the cited software software,
4. that the identifiers and metadata about software should be persistent,
5. that citations should enable access to the software and associated information about the software that informs its use, and
6. that citations should facilitate identification of and access to the specific version of software that was used, such as by version number, revision numbers, or variants such as platforms.

The talk also provided practical information on how to semi-automatically take code on GitHub and publish it on Zenodo, obtaining a DOI that can then be cited.[16] Finally, the talk brought up a number of the ongoing discussions at the WSSSPE and Force11 working groups and their determinations, such as what software to cite, how to uniquely identify software, that peer-review of software is important but not required for citation, and how publishers can help.

References

1   Arfon M. Smith, Daniel S. Katz, Kyle E. Niemeyer, and FORCE11 Software Citation Working Group. Software citation principles. *PeerJ Computer Science*, 2:e86, September 2016.

3.11   Best Practices by Any Other Name

Katie Kuksenok (University of Washington – Seattle, USA)

This talk looked at intersections of the technical, social, and cognitive aspects of software engineering in research, and asked how the available community and skill resources could be leveraged. It brought together various elements raised throughout the workshop so far, including different roles that had been identified, the need for software engineers to learn from scientists just as we hope researchers learn software engineering practices, and overcoming communications barriers.

Kuksenok also provided a link to a short blog post summary with figures.[17]

---

[16] https://guides.github.com/activities/citable-code/
[17] Shortened URL to article at https://medium.com/hci-design-at-uw: https://goo.gl/Lxqrt5.

3.12 ASCL: Restoring Reproducibility – Making Scientist Software Discoverable

Alice Allen (University of Maryland – College Park, USA)

Allen presented an overview of the Astrophysics Source Code Library (ASCL)[18] and its history. She also discussed a few of the changes to the ASCL infrastructure, lessons learned from looking at what other astro code registries and repositories had done, what ASCL did with those lessons, and some of the impact ASCL has had on the community.

3.13 A Short (and Probably Incomplete) History of Research Software Engineers in the UK

Robert Haines (The University of Manchester, GB)

"*Before software can be reusable it first has to be usable*" – Ralph Johnson, University of Illinois at Urbana-Champaign.

A growing number of people in academia combine expertise in programming with an intricate understanding of research. Although this combination of skills is extremely valuable, these people lack a formal place in the academic system; they are not academics with a personal research agenda. This means there is no easy way to recognize their contribution, to reward them, or to represent their views.

One of the largest obstacles to overcome in recognizing this group of people is that they are often "hiding" in their institutions under a myriad different job titles and roles: Post-Doc, Research Associate, System Administrator, Computer Officer, and so on. In the instance of Post-Docs and Research Associates it is often the case that these people suddenly find that they have written too much code, and not enough papers, and so they fall foul of the usual metrics used to evaluate them for promotion. Being the person in the lab who "knows about computers" can be detrimental to your career.

These topics came up frequently at the Software Sustainability Institute's Collaborations Workshop in 2012. At this "unconference" style event a number of us repeatedly found ourselves in sessions discussing career paths, credit, recognition, metrics and reward, for those of us working in academia, who weren't academics. Without a name, it is difficult for people to rally around a cause, so we created the term Research Software Engineer (RSE) to describe the intersection of "The Craftsperson and the Scholar".[19] RSEs are facilitative, supportive and collaborative; part of the academic community and its institutional memory, providing continuity and stability for its academic software. We also created the UK Community of Research Software Engineers[20] (UKRSE) as a focal point for our future campaigns and the

---

[18] http://ascl.net
[19] http://www.software.ac.uk/blog/2012-11-09-craftsperson-and-scholar
[20] http://www.rse.ac.uk/

Institute made the promotion of the RSE job role a cornerstone of their policy, lobbying the UK Government and Research Councils for RSEs to be recognized at a high level.

Since 2012 the RSE job role has gained traction in a number of institutions and has been endorsed by the Engineering and Physical Sciences Research Council in the UK (EPSRC). There are groups employing RSEs in University College London, the University of Manchester, the University of Cambridge, the University of Southampton and the University of Sheffield, with more in the process of being set up all the time. The EPSRC has funded seven RSE Fellowships, who have a remit to develop and support software and users of software, and has also funded a network of RSE leaders[21] to further build the community of RSEs and develop the next round of Fellowships.

This year we held the world's first RSE Conference in Manchester, UK.[22] It was the first conference to focus on the practice of research software engineering and included sessions on the issues that affect people who write and use software in research as well as presented talks and workshops where new tools and techniques were taught. The conference attracted over 200 people from 14 countries, so we look forward to further expanding our community and building links with colleagues all over the world in the near future.

3.14   101companies – Making a Failing Project Succeed

Ralf Lämmel (Universität Koblenz-Landau, DE)

Lämmel presented the 101Companies project: a software *chrestomathy*, from "chresto", meaning "useful" and "mathein", meaning "to learn". 101 is a knowledge resource for technological space travel (between all kinds of technological spaces). It can serve to compare technologies, for programming education, and can serve as a playground for student projects. Lämmel discussed some of the challenges the project is experiencing and some of the ways in which it is succeeding.

3.15   UW eScience Institute Initiatives

Cecilia Aragon (University of Washington – Seattle, USA)

Thanks in part to the recent popularity of the buzzword "big data," it is now generally understood that many important scientific breakthroughs are made by interdisciplinary collaborations of scientists working in geographically distributed locations, producing and analyzing vast and complex data sets. The extraordinary advances in our ability to acquire and generate data in physical, biological, and social sciences are transforming the fundamental nature of science discovery across domains. Much of the research in this area, which has become known as data science or eScience, has focused on automated methods of analyzing

---

[21] http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/N028902/1
[22] https://www.software.ac.uk/blog/2016-10-04-future-rses-looking-rosy-following-phenomenal-conference

data such as machine learning and new database techniques. Less attention has been directed to the human aspects of data science, including how to build interactive tools that maximize scientific creativity and human insight, and how to train, support, motivate, and retain the individuals with the necessary skills to produce the next generation of scientific discoveries.

In this talk, Aragon discussed the history and ongoing initiatives at the UW eScience Institute, including opportunities to participate in the $37.8M Moore/Sloan Data Science Environment at UW, UCB, and NYU, and speculate upon future directions for data science. In particular, she discussed new initiatives at UW such as the eScience Incubator and the Data Science for Social Good program and focused on results of ethnographic studies of their projects and future work in the Data Science Studies working group. She further argued for the importance of a human-centered approach to data science as necessary for the success of 21st century scientific discovery. She attested that we need to go beyond well-designed user interfaces for data science software tools to consider the entire ecosystem of software development and use: we need to study scientific collaborations interacting with technology as socio-technical systems, where both computer science and social science approaches are interwoven.

3.16 The Netherlands eScience Center

Rob van Nieuwpoort (The Netherlands eScience Center (NLeSC), NL)

The Netherlands eScience Center (NLeSC) is the Dutch national hub for the development and application of domain overarching software and methods for the scientific community. NLeSC develops crucial bridges between increasingly complex modern e-infrastructures and the growing demands and ambitions of scientists from across all disciplines. The application of digitally enhanced scientific practices makes sure that returns can be achieved from scientific investments. In support of this goal NLeSC funds and simultaneously funds and participates in multidisciplinary projects, with academia and industry, with optimized data-handling, efficient computing and big-data analytics at their core.

NLeSC contributes exclusively to multidisciplinary projects with the potential to deliver scientific excellence, in terms of breakthroughs and in the realization of unique eScience methodologies. Many organizational practices, such as our open call strategy and other funding models ensure that new projects fulfill these criteria.

Apart from contributions to scientific publications in high-impact scientific journals and conferences, NLeSC's primary deliverables are eScience instruments (*e.g.*, software tools, workflows). Whilst the instruments may include a domain specific component, primarily these tools overarch multiple domains. The instruments are efficient, calibrated, reliable and accessible, and based on excellent standards of code quality utilizing meta-data standards and software development environments. Successful instruments are made publicly available as part of NLeSC's eScience technology platform (eStep) program.[23] This platform provides easy access to the developed tools and instruments to the broader scientific community and industry alike. NLeSC also shares non-scientific technical results, documentation and best practices in the knowledge base that also is a part of eStep.

---

[23] See also http://estep.esciencecenter.nl

NLeSC also plays a key role in optimizing and disseminating the best practices in the areas of software sustainability and data-stewardship, including the need to engage with communities of practices, data-publishers and related initiatives.

The rapid growth of data and computing initiatives risks unnecessary fragmentation and duplication. NLeSC works with numerous partner organizations, nationally and internationally, to identify common challenges such as training and career support for eScientists, as well as providing thought leadership on issues such as data-stewardship and software sustainability. NLeSC is a joint initiative of the Dutch national research council (NWO) and the Dutch organization for ICT in education and research (SURF).

3.17 On ImpactStory, Scientific Software Map, and depsy

James Howison (University of Texas – Austin, USA)

There is a need to provide insights into the scientific software ecosystem [1]: the set of projects, their software products, their authors, their dependencies, the papers describing them, and papers that have used the software to undertake science. Such insights are useful for many players in the ecosystem, including end-users, software component producers, and ecosystem stewards (funders and senior scientists). Howison presented two systems that have attempted to gather and display this data: depsy.org[24] and the scientific software network map.

Depsy has gathered data from CRAN and PyPI as a starting point. They gather dependency and authorship information from those repositories. They identify the software in the literature using a fulltext keyword search for the name of the package. They are then able to calculate both direct mentions of each package and indirect mentions, using the PageRank algorithm. Depsy is produced by ImpactStory (Heather Piowawar and Jason Priem) [2].

The second system presented was the Scientific Software Network Map by Bogart, Howison, and Herbsleb.[25] The Map is designed to be populated from different ecosystems' software repositories, current work including data from two locations: R scripts on GitHub, and data from the Texas Advanced Computing Center gathered about jobs submitted to their supercomputing infrastructure. The interface uses D3 for the visualizations, and Pyramid, Mongo and Jinja for the web and database framework. Maps are designed to directly address the needs of scientific software producers and stewards for usage-related information about packages. The tool's features include a usage graph over time, a filterable/sortable list of packages, a "co-usage" graph showing what packages were used together, and a listing of external software (*e.g.*, end-user scripts and packages under development) that depend on each package. The co-usage graph could be used to identify previously unknown clusters of packages and to bring their developers together.

References
1   Chris Bogart, James Howison, and James D. Herbsleb. Mapping the Network of Scientific Software. page (Abstract Submission), Washington, D.C., June 2015.
2   Dalmeet Singh Chawla. The unsung heroes of scientific software. *Nature*, 529(7584):115–116, January 2016.

---

[24] http://depsy.org
[25] http://scisoft-net-map.isri.cmu.edu:7777

3.18   The OSSMETER platform

Jurgen J. Vinju (CWI – Amsterdam, NL)

Vinju briefly introduced the OSSMETER platform[26] for monitoring and comparing open-source software projects in terms of code, activity, issues and community. This open-source project can be used to monitor and assess projects and may be applicable to the domain of scientific software as well.

4   Breakout sessions

The afternoons of the workshop were dedicated to focused break-out groups. The groups were defined in a plenary discussion using a board with sticky notes. Everybody could propose topics. The topics were grouped on the board by topic similarity. Some groups continued over more than one day in order to arrive at a tangible result. All groups made notes into a single shared document. This same document was the source for the current report as well as the manifesto.

The breakout groups are detailed below in arbitrary order.

4.1   Research Software Project Typology

Benoit Combemale, Jurgen J. Vinju, Robert Hirschfeld, Ralf Lämmel, Daniel Garijo, Christoph Becker, Caroline Jay, Robert Haines, and Cecilia Aragon

The goal of this group was to characterize academic software projects across different dimensions (*i.e.*, motivation, impact, costs, risks, strengths and weaknesses) so as to analyze their impact in (i) the creation of software outputs, (ii) the acquisition of new funding and (iii) the recognition of adequate author credit. Ideally, this characterization would be structured as an ontology, extending existing software metadata vocabularies,[27] and specializing them for academic software.

The group paid particular attention to the different roles of academic software engineers and researchers in the context of their software projects. This discussion led to the creation of the "Software Credit Role" ontology,[28] which identifies the types of contributions of academic software at any stage of the software development (see Figure 1 for some examples). The ontology will soon extend DOAP[29] and schema.org,[30] and will be mapped to the crosswalk list CodeMeta terms.[31]

---

[26] http://www.ossmeter.org

[27] For example http://ontosoft.org/ontology/software/

[28] https://w3id.org/softwareCredit

[29] http://usefulinc.com/ns/doap#

[30] http://schema.org/

[31] https://github.com/codemeta/codemeta/blob/master/crosswalk.csv

Figure 1 Excerpt from https://w3id.org/softwareCredit depicting the Person concept with the different roles a person could take upon themselves.
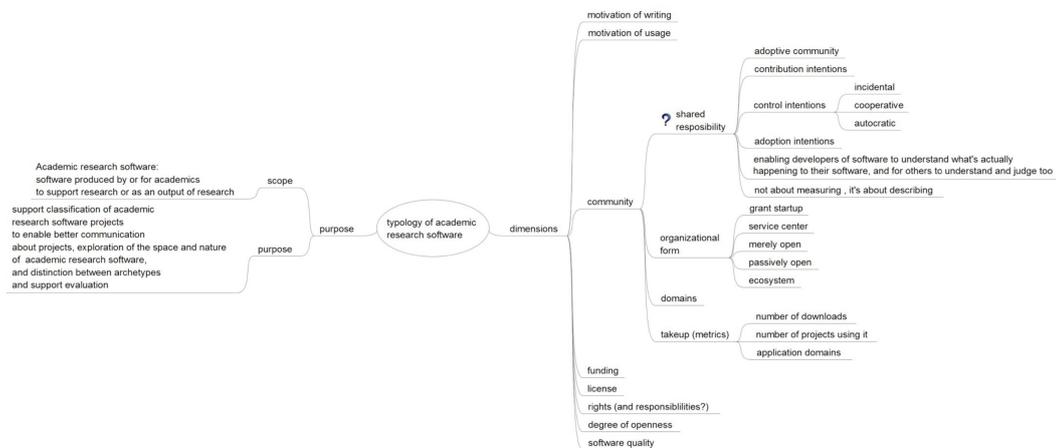
Figure 2 A mindmap of a brainstorm towards an academic software project typology.

The group brainstormed about the project typology in three sessions. The first session led to divisions along the lines of the following concepts: intentions of usage, degree of openness, funding, contributors (single person, group, closed/open community), community (scale, culture, rights, license), maturity (process), accessibility/openness and organizational openness from engineering properties (code quality, test coverage, benchmarking, release management, semantic versioning, etc.).

The second brainstorm refined the outcome of the first one in the mindmap depicted in Figure 2.

The third discussion produced a faceted classification with yet different distinctions mainly focused on purpose, intentions and motivations. This finally resulted in a questionnaire (shaped as an online spreadsheet[32]) which we filled in for a few dozen projects to see if the distinctions would generate a meaningful overview of typical academic software projects.

The group discussions produced enough material to start working on a publication about the topic. We believe that it is important to understand, describe, analyze and improve academic software projects descriptions, as clear classifications of software projects are not currently commonplace.

---

[32] https://goo.gl/ud5Ik2

4.2   Empirical Study of Software in Conferences

Jeffrey Carver, James Howison, Robert Haines, Caroline Jay, Kevin Crowston, and Oscar Nierstrasz

The goal of this group was to explore the possibility of carrying out an empirical study to better understand *what software is cited in academic conferences in a particular domain*, how such software is cited, and what are common software practices in that domain.

The group mainly focused on identifying (i) potential *research questions* to be addressed, and (ii) *procedural questions* concerning the logistics of carrying out such a study.

Some of the specific research questions considered were:

- Which software ends up being mentioned in papers?
- Who are the developers? (PhD students? Research Software Engineers?)
- What happens to software after the paper is published?
- What software engineering practices are applied? (Version control? Testing?)
- Who pays for software development?
- What problematic issues commonly arise?
- What recommendations would improve the quality of software in the fields?

Procedural questions included:

- How to achieve variance? (Do you code for research questions? Venues? Should a grounded approach be used or should predefined hypotheses be considered?)
- Would machine learning help to classify software in papers?
- How to start? (Which domain to select?)

Katy Huff notes that the Berkeley Institute for Data Science (BIDS, with collaboration from the UW eScience Institute and the NYU center for data science) has collected over 30 case studies from scientists from various domains, synthesized them into lessons learned, and compiled them into a book to be published in 2016 by the University of California Press.[33]

4.3   Examining Sustainability for a Particular Project

Carole Goble, Katie Kuksenok, Christoph Becker, Daniel Garijo, Mike Croucher, and Daniel S. Katz

Sustainability refers to the capacity of a system to endure over time. In the case of software this means that it must continue to be available in the future, on new platforms and meeting new needs.

This breakout group discussed various aspects of sustainability of scientific software: How can we ensure sustainability of scientific software? What does this mean for a particular

---

[33] https://github.com/BIDS/repro-case-studies

project? Does it make sense to talk about sustainability of software divorced from the sustainability of the team?

To produce sustainable software requires sustaining the project organization – or at least "a" project organization – which produces and maintains the software. With a project we mean an (evolving) team of people and their organization. Sustainability of the software and of an organized group of people who take responsibility for it are clearly entangled, since you can say very little about the technical sustainability of a software system alone without considering the social system. Note that a sustainable team does not imply an immutable team: a sustainable team can grow or shrink, distribute and swap out team members or leadership in the long term.

A key factor related to software sustainability is *technical debt*, *i.e.*, technical shortcuts to achieve a short-term goal that impact long-term quality and maintainability. An example of investment needed to pay off this debt is the effort required to make the code more modular and more amenable to accepting new plug-in components. A challenging scenario would be to conduct an analysis of sustainability debt for specific systems in a short time since there are multiple, interlinked systems, services, communities, and stakeholder groups. A recurring problem is the issue of obtaining funding to maintain scientific software and guarantee its sustainability for the community and the long tail of users. Large projects, funders, publishers, institutions all have an obligation or a role to support (subsidize) the software and the service.

A number of software maturity frameworks were discussed such as the Software Sustainability Maturity Model (SSMM), OSS Watch Openness Rating, NASA Reuse Readiness Rating, CMM, and QSOS. User-producer reciprocity misalignments were discussed leading to an accepted lightning talk at WSSSPE 2016 which credited the seminar (C. Goble (2016), A Simple Profiling Framework for Software User-Producer Reciprocity Review, presented at 4th WSSSPE (Workshop of Sustainability of Science Software Practice and Experience), 2016).

4.4   Making the Impact of Software more Visible

Matthew Vaughn, Katy Huff, Matt Turk, Rob van Nieuwpoort, Alice Allen, Andrei Chiş, Cecilia Aragon, Claude Kirchner, and Daniel S. Katz

A fundamental problem in selected scientific fields, such as physics and astronomy, is that a huge amount of effort is invested in producing scientific software, but only published papers count towards scientists' careers. This leads inevitably to the process, "I write the software and then I write a paper to get credit." We need a cultural change in the scientific community to raise awareness that scientific software itself represents valuable intellectual content and scientific innovation directly and explicitly.

This breakout group discussed the status quo and ways to change it. Journals exist in Computer Science and Bio-Informatics that are home to intellectual contributions of software packages. Several computer science conferences explicitly support "artifact evaluation" where software contributions are scrutinized and appreciated as part of the academic review process. But the appreciation of software as academic output does not seem to be universal across domains.

Appreciation for software seems more natural in Computer Science, which explains recent efforts in this field to making the impact of software visible, yet across the board the field is not particularly ahead in this regard with respect to other fields. An important positive factor is the recent focus in research towards "innovation" and "valorization" which puts tangible and transferable results of Computer Science in the form of working software in the spotlight. Bio-Informatics though, is ahead of the pack, which may be due to its explicit branding as the informatics branch of Biology.

One critical need that was identified is to explicitly include software output into the evaluation processes of both academics and academic institutions. Currently, publishing software may give you a DOI (*e.g.*, via Zenodo[34]) but this neither brings you reputation nor does it contribute to a positive evaluation per se, unless this DOI is indexed as part of your records and citations to it are credited to your metrics. Problems with making software output part of the academic process were discussed. For example, software lasts longer than the citable snapshots we have now, leading to evolving author lists and evolution between software versions, which may in turn lead to software growing far beyond the original scope. Short papers about software are too short to make the academic challenges and contributions observable, while long papers about software hide the software contributions under the traditional paper-style contributions. And finally, reading and appreciating source code is just hard especially if it is not originally written with such an audience in mind.

Several possible solutions were discussed:

- Papers could capture more about the software's intellectual impact; *i.e.*, in a separate section akin to the standard "research method" and "threats to validity" sections.
- Letters of support may be the best way to communicate software impact to tenure committees.[35]
- People could claim their contributions more explicitly: web pages are needed to document software contributions; research software engineers should be motivated to co-author papers; and scientists who develop software must be willing to be proud and certainly unapologetic about their software contributions in front of tenure committees.
- Rubrics and templates for letters of recommendation should be developed to highlight intellectual contributions from code.
- A Research Coordination Network (RCN) workshop[36] should be started to showcase intellectual content for academic software.
- Encourage the formation of a prestigious scientific software award.

References

1   Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council. *Academic Careers for Experimental Computer Scientists and Engineers Committee on Academic Careers for Experimental Computer Scientists.* National Academy Press, 1994.

---

[34] https://zenodo.org/

[35] This has been the case in Computer Science in the US for about 20 years, due to a National Research Council report [1] that is frequently quoted in such letters.

[36] RCNs are an NSF instrument: https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=11691. Presumably similar instruments exist in other countries.

4.5   Reviewing FORCE11 Software Citation Principles

Dan Katz, Robert Haines, James Howison, Katy Huff, Caroline Jay, and Matt Vaughn

This group came together to contribute to the FORCE11 software citation principles. The FORCE11 website[37] says: "Based on a review of existing community practices, the goal of [FORCE11] was to produce a consolidated set of citation principles that may encourage broad adoption of a consistent policy for software citation across disciplines and venues."

The breakout group reviewed the different personas (roles) in relation to the software citation principles to see if their interests were reflected appropriately: reviewer, author of software, designer of citation style, reader of paper, user of software, funding agency, publisher.

This review was passed on to the FORCE11 Software Citation Working Group editors, leading to updates to the citation principles where these were deemed necessary, and the final version was recently published [1], including these updates.

References
1   Arfon M. Smith, Daniel S. Katz, Kyle E. Niemeyer, and FORCE11 Software Citation Working Group. Software citation principles. *PeerJ Computer Science*, 2:e86, September 2016.

4.6   Research Software Engineering Handbook

Jeff Carver, Mike Croucher, Andrei Chiş, Katie Kuksenok, Rob van Nieuwpoort, Kevin Crowston, Robert Haines, and Katy Huff (partial attendance)

This group focused on designing a handbook with practical advice for academic research personnel involved in programming, or management of, a programming project. This audience includes RSEs, PIs, research scientists and graduate students. Though the particular projects and needs vary widely among these people, the breakout group identified a common need for a resource to support the selection and integration of tools and skills from an overwhelmingly wide array of available options. The experiences of Mike Croucher and Rob van Nieuwpoort in supporting scientific groups grounded the discussion, and the concept of the handbook arose from a similar living document developed within van Niewpoort's group.[38]

Many resources exist for learning and improving programming and management skills. Rather than duplicating these existing resources, the handbook will curate and describe them, helping to select and integrate distinct tools and skills through its need-centered organization. The group discussed how to accommodate the wide range of subjects and audiences. The outcome of this discussion was a table of contents with high-level sections articulated around the outstanding need of the reader:

---

[37] https://www.force11.org/software-citation-principles
[38] https://nlesc.gitbooks.io/guide/content/

- **Getting started** covers the basics of version control, licenses, citing software, looking for existing solutions, and getting feedback.
- **"I want to code better."** Provides more information on testing, continuous integration, release management, documentation, and other skills that help a programmer to produce higher-quality code.
- **"I want better code."** Focuses on properties of the code produced, not just the approach to production, with the intent of writing more reusable and testable software.
- **"I have a team of developers now. How can I cope with the new challenges ahead?"** Offers additional resources on management and communication processes, formal requirements gathering, and other issues related to moving toward team-based development.
- **"I have my first pull request, and the beginning of a community. What now?"** Returns in more depth to issues of licensing and release management, with added information about community-building and sustainability concerns.

Each section contains 6–10 chapters. In addition to the overall structure, the group discussed an index to tailor recommended chapters to team size (*e.g.*, a graduate student and a PI vs. a 5-person team) and role (*e.g.*, the student who wants to improve vs. the PI who wants to implement better team processes). As an outcome of the breakout sessions, the group refined a detailed table of contents, assigned contributors to each section, and created a GitHub repository to help manage this document.[39]

4.7   Future Research directions

Claude Kirchner, James Williams, Oscar Nierstrasz, Katie Kuksenok, Jurgen J. Vinju, Benoit Combemale, Matt Vaughn, Cecilia Aragon, and Alice Allen

This breakout group discussed a number of possible themes for future research projects or initiatives. Of the topics discussed the following ones were elaborated in enough detail to be included as part of the Dagstuhl Manifesto:

1. *Quantifying the availability of scientific software:* this research would attempt to determine how research software exists, for which domains is it developed, who owns it, who pays for it, who maintains it. The research would also attempt to identify opportunities for reuse, sharing, and collaboration.
2. *Facilitating software discovery within and across disciplines:* the key research question here is to determine, aside from standards, what other approaches can effectively support discoverability of available research software.
3. *Sustainability of software experimentation:* how can we ensure reproducibility of software experiments, not just in the short or medium term, but in the very long term?
4. *Software engineering tools improving productivity by tailoring to intent and skill:* not all research software requires the same development rigor and discipline as commercial software, yet all would benefit from from skills beyond that of "hobbyist" programmers.

---

[39] https://github.com/DagstuhlEAS/rse-handbook

How can we appropriately select and adapt the level of software engineering discipline appropriate to a given research project?

5. *Re-tooling the bibliographic software toolchain for software citation:* existing bibliographic tools are highly tailored towards academic writing. How do we adapt the metadata and tools to get the right meta-information about software into citations?

6. *Analysis of scientific software ecosystem metadata:* how can we effectively monitor and analyze metadata about research software in the large?

4.8 Design of the Manifesto

Claude Kirchner, Oscar Nierstrasz, James Howison, Katie Kuksenok, and Jurgen J. Vinju

A key goal of this Dagstuhl Perspectives Workshop has been to produce a *Dagstuhl Manifesto*[40] to serve as a roadmap towards future professional software engineering for software-based research instruments and other software produced and used in an academic context.

Throughout the week, each of the breakout groups logged their discussions in a common, shared Google document, and took care to identify specific "pledges" that could serve as input to the manifesto. The goal was to ensure actionable outcomes which called on individuals themselves to act, rather than to ask others to act on their behalf. This intent acknowledges that "we" are the community and that visible action provides "social proof" and motivates others. The pledges were based on a template: (i) the pledge itself, expressed in the form "I will *take some specific actions to support EAS*", (ii) *background* motivating the pledge, (iii) *contradictions or concerns* that could impact the pledge, (iv) specific *actions needed*, and (v) identification of other *players who need to act*.

By the end of the week, some 30-odd candidate pledges had been collected for the manifesto. The manifesto design breakout group then set out to cluster the pledges around common, overarching themes, namely:

 (i) Citation & Reviewing;
 (ii) Recognition;
 (iii) Making intellectual content visible;
 (iv) Software Projects; and
 (v) Sustainability.

In a subsequent session, the group pared down the list of candidates, eliminating seemingly redundant or confusing pledges, and pledges lacking substantial description, yielding 18 surviving candidates.

Subsequent to the termination of the workshop, a poll was prepared to rank these 18 pledges. The list was subsequently simplified and reduced to 5 key pledges in three categories: (i) Citation, (ii) Careers, and (iii) Development. (The poll has also been run at WSSSPE 2016, though the results have not yet been analyzed.)

At the time of the preparation of this report, a draft manifesto is under preparation as a GitHub project.[41]

---

[40] Dagstuhl Perspectives Workshops explore new and emerging topics in Computer Science, and are expected to produce *Dagstuhl Manifestos* that capture trends and developments related to these topics. See: http://www.dagstuhl.de/en/publications/dagstuhl-manifestos/

[41] https://github.com/DagstuhlEAS/draft-manifesto

4.9   Shoot the Dogma and War Stories

Carole Goble (convenor), Kevin Crowston, Jurgen J. Vinju, Alice Allen, Robert Haines, Caroline Jay, Mike Croucher, James Howison, and Katy Huff (partial attendance)

In this relaxed evening session participants let off steam regarding dogmas – a principle or set of principles laid down by an authority as incontrovertibly true – that needed to be challenged, and to relate war stories. Ten dogmas and two war stories were discussed:

1. *Use the best language for the job:* Using whatever language is right for the various components of a system, thinking that component making is "fun" rather than thinking about the total system. The result is build misery and a tower of pain. It is better to have a language that is good enough and fits/the same as the languages you are using than use another language. If the build has taken over the whole development it has become a monster.

2. *Blockchain solves everything:* Security is a life's work and best left to specialists.

3. *C++ is necessary for high performant code:* This is not true and lazy thinking. C is good for parallel computing and the libraries, but keep away from the bells and whistles, and the "++" part of C++.

4. *A code repository gives perfect tracking:* The repository trace is the "farts of dinosaurs" – that is it records what happened long ago and often not the right thing. Histories can be meddled with.

5. *Writing documents using Git is good:* Git is an excellent software writing environment. It is not a document narrative writing environment. Starting up with independent section development has a place, but for one voice narrative it is poor and a sure way to put off collaborators.

6. *Python solves everything:* No-one shot this Dogma. So perhaps it does solve everything.

7. *Docker solves everything:* Docker solves a specific problem, but we should recognise that build systems and dependency management are hard. Silver bullets are just bullets that are useful.

8. *Engineers know best how to estimate effort:* Engineers do not. They have a tendency to be optimistic and want to please, impress, over state their ability, understate complexity and underestimate the available time and the absence of distractions or obstacles. A rule of thumb is to double the unit and raise the unit of estimates (*e.g.*, 1 day = 2 weeks, 1 week = 2 months).

9. *Compute privacy is necessary on shared computing resources:* This is tool privacy, not data privacy. Running tools in secret is unnecessary and, to show their impact, tool usage needs to be tracked.

10. *Agile solves everything:* Agile is a religious fervour that can be considered harmful (thoughtless evangelism is suspicious). Agile is sometimes used as a PRINCE 2.0 backlash. Processes themselves have life and agile can work when thoughtfully used. Studies indicate that using KANBAN (a sub-process of Agile) is better than no set process or a full agile process.

Two war stories were shared:

1. *Frankenstein technology stacks:* A common tale is the legacy technology stack using software often developed by non-software engineers. The result is a "walk of shame" for the software engineers: users are proud and even brag about the software but developers

would rather it was kept quiet. Conclusion: compromise is commonplace but be sure to clarify the impact on the participants.

2. *Wasting participants time:* A tale about starting a citizen science project that gained participants but no resources had been planned when it got going, leading to frustration and wasting the time of the citizens. Conclusion: think through the consequences of success before starting projects that recruit participants.

5 Acknowledgements

Participants

- Alice Allen
  University of Maryland –
  College Park, US

- Cecilia Aragon
  University of Washington –
  Seattle, US

- Christoph Becker
  University of Toronto, CA

- Jeffrey Carver
  University of Alabama, US

- Andrei Chis
  Universität Bern, CH

- Benoit Combemale
  IRISA – Rennes, FR

- Mike Croucher
  Software Sustainability Institute –
  Edinburgh, GB

- Kevin Crowston
  Syracuse University, US

- Daniel Garijo
  Technical Univ. of Madrid, ES

- Ashish Gehani
  SRI – Menlo Park, US

- Carole Goble
  The Univ. of Manchester, GB

- Robert Haines
  The Univ. of Manchester, GB

- Robert Hirschfeld
  Hasso-Plattner-Institut –
  Potsdam, DE

- James Howison
  University of Texas – Austin, US

- Katy Huff
  University of California –
  Berkeley, US

- Caroline Jay
  The Univ. of Manchester, GB

- Daniel S. Katz
  University of Illinois at Urbana
  Champaign, US

- Claude Kirchner
  Inria – Le Chesnay, FR

- Katie Kuksenok
  University of Washington –
  Seattle, US

- Ralf Lämmel
  Universität Koblenz-Landau, DE

- Oscar M. Nierstrasz
  Universität Bern, CH

- Matthew J. Turk
  University of Illinois at Urbana
  Champaign, US

- Rob van Nieuwpoort
  VU University Amsterdam, NL

- Matthew Vaughn
  University of Texas – Austin, US

- Jurgen J. Vinju
  CWI – Amsterdam, NL