AGU PUBLICATIONS

Earth and Space Science

RESEARCH ARTICLE
10.1002/2016EA000225

Special Section:
Geoscience Papers of the Future

Key Points:
- Geoscientists are increasingly involved in developing specialized scientific software for research
- There is no standard method for assigning credit through citation of software, and practices vary
- A survey of current practices in geodynamics reflects the community seeks to cite and does cite software at a high rate

Supporting Information:
- Supporting Information S1

Correspondence to:
L. J. Hwang,
ljhwang@ucdavis.edu

Citation:
Hwang, L. J., Fish, A., Soito, L., Smith, M., & Kellogg, L. H. (2017). Software and the scientist: Coding and citation practices in geodynamics. *Earth and Space Science*, *4*, 670–680. https://doi.org/10.1002/2016EA000225

Received 30 SEP 2016
Accepted 2 OCT 2017
Accepted article online 11 OCT 2017
Published online 12 NOV 2017

Software and the Scientist: Coding and Citation Practices in Geodynamics

Lorraine Hwang[1], **Allison Fish**[2,3,4], **Laura Soito**[2,5], **MacKenzie Smith**[2], and **Louise H. Kellogg**[1]

[1]Department of Earth and Planetary Sciences, University of California, Davis, CA, USA, [2]University Library, University of California, Davis, CA, USA, [3]School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA, [4]TC Beirne School of Law, University of Queensland, Brisbane, Queensland, Australia, [5]College of University Libraries & Learning Sciences, University of New Mexico, Albuquerque, NM, USA

Abstract In geodynamics as in other scientific areas, computation has become a core component of research, complementing field observation, laboratory analysis, experiment, and theory. Computational tools for data analysis, mapping, visualization, modeling, and simulation are essential for all aspects of the scientific workflow. Specialized scientific software is often developed by geodynamicists for their own use, and this effort represents a distinctive intellectual contribution. Drawing on a geodynamics community that focuses on developing and disseminating scientific software, we assess the current practices of software development and attribution, as well as attitudes about the need and best practices for software citation. We analyzed publications by participants in the Computational Infrastructure for Geodynamics and conducted mixed method surveys of the solid earth geophysics community. From this we learned that coding skills are typically learned informally. Participants considered good code as trusted, reusable, readable, and not overly complex and considered a good coder as one that participates in the community in an open and reasonable manor contributing to both long- and short-term community projects. Participants strongly supported citing software reflected by the high rate a software package was named in the literature and the high rate of citations in the references. However, lacking are clear instructions from developers on how to cite and education of users on what to cite. In addition, citations did not always lead to discoverability of the resource. A unique identifier to the software package itself, community education, and citation tools would contribute to better attribution practices.

1. Introduction

Through the continuous cycle of observation and integration of new knowledge, scientific discovery historically relied on two pillars—hypothesis and experimentation to open new pathways for acquisition of knowledge and betterment of all. In the natural world, carefully constructed experiments involve observations. Increasingly, computation has become a new tool for experimentation to a degree that computational science has been called the third pillar of science (Reed et al., 2005).

Scientists today rely on computers and the associated software tools for most aspects of their investigations including data collection, storage, processing, analysis, and simulation (Hannay et al., 2009). Numerical modeling, mathematical modeling of time dependent behavior, itself has become an essential tool in design and manufacturing and also in science to study processes that often are impractical, inexpensive, and/or impossible to observe in the real world. For research on the Earth's interior and other large-scale processes, for example, it is impractical or impossible to encapsulate a planet or a continent in a traditional laboratory setting.

In the geosciences, more than 60% of graduate students use computer-based methods (C. Wilson, 2014), on par with other research methods such as literature-based, field-based, and lab-based methods. The U.S. geoscience community, including remote sensing, has for several decades collected, developed, archived, and maintained large data sets and data products through organizations such as the National Center for Atmospheric Research, National Atmospheric Science Administration, National Oceanographic and Atmospheric Administration, U.S. Geological Survey, Incorporated Research Institutions for Seismology (IRIS) (Trabant et al., 2012), and UNAVCO; through database projects such PetDB for petrological data (Lehnert et al., 2000); and at specialized or regional data centers (Romanowicz et al., 1994). These data have driven the development of software tools for data processing and manipulation and act as drivers in the trend toward computation as researchers incorporate large data sets into their studies.

Table 1
2015 CIG Codes

Name	Support level
Short-term crustal dynamics	
PyLith	D_CIG
RELAX	D_CONTRIB
VirtualQuake	D_CONTRIB
SELEN	S_CONTRIB
LithoMop	A
Long-term tectonics	
Gale	A
Plasti	A
SNAC	A
Mantle convection	
ASPECT	D_CIG
CitcomCU	D_CONTRIB
CitcomS	D_CONTRIB
ConMan	S_CONTRIB
Ellipsis3d	A
HC	A
Seismology	
Axisem	D_CONTRIB
Burnman	D_CONTRIB
Mineos	A
Flexwin	A
Seismic CPML	S_CONTRIB
SPECFEM3D	D_CIG
SPECFEM3D Globe	D_CIG
SPECFEM3D Geotech	D_CONTRIB
SPECFEM2D	D_CONTRIB
SPECFEM1D	S_CONTRIB
SW4	D_CONTRIB
Geodynamo	
Calypso	D_CIG
MAG	A
Computational science	
Cigma	A
Exchanger	A
Nemesis	S_CONTRIB
Pythia	S_CONTRIB

Note. A: Archived. D_CIG: Developed with support from CIG. D_CONTRIB: Developed by contributors to CIG. S_CONTRIB: Supported by contributors to CIG.

Significant development of scientific software is also required to build the computational capacity for modeling these data, including analysis and visualization and forward modeling and simulation. The effort of writing, documenting, disseminating, and maintaining such software in a research environment poses challenges that share some characteristics from curating data sets. Like data sets, the mechanisms for publishing and assigning attribution and citation are not yet standardized and are unevenly used. While technical reports and, more recently, data publications may be written about a data set, specialized scientific software is more complicated. Scientific software development efforts may result in a release announcement in a newsletter such as *EOS* (e.g., Wessel et al., 2013) and/or a citable publication about the algorithm (e.g., King et al., 1990). The latter more commonly focuses on scientific results found using computational models than on the software or algorithms themselves. Publications about scientific software itself are not common but increasing with targeted journals such as the *Archive of Numerical Software*, the *Electronic Seismologist* column of the *Seismological Research Letters*, and *Journal of Open Source Software*. A few broader domain journals, such as *Geoscientific Model Development* and *Computers and Geoscience* (Chue Hong, 2014), have been publishing papers about geoscience software since 2009 and 1975 respectively. However, often missing from publications about the development of research software are secondary processes such as quality checks that vet the implementation of the software through code review, benchmarking, and testing (automatic, unit, and regression) which is just as important as the review of its theoretical foundation (Kelly et al., 2009; Merali, 2010; Oberkampf & Roy, 2010).

The development of scientific research software for specialized modeling and data analysis represents a significant intellectual contribution. Yet the standard means of attributing scholarly credit and provenance, citation of a publication, do not readily adapt to scientific software, and no standard has emerged for citing software. Here we examine the social and technical barriers to software citation in the geodynamics community by mixed method surveys and examining the publication record and software practices in the Computational Infrastructure in Geodynamics (CIG) community. We begin by describing this community of practice, how individuals learn to write scientific software, and the community's thoughts surrounding coding practices. This sets the context to discuss the reward system in science and the efforts to elevate software. We then look at efforts of the CIG community to give credit within their community, preserve provenance, and promote reproducibility, and we examine the outstanding issues in giving credit for software.

2. CIG as a Community of Practice

The Computational Infrastructure for Geodynamics (CIG), geodynamics.org, is a specialized community of practice (Lave & Wenger, 1991) within the geosciences. The community formally and informally learns from each other the best practices in the usage and development of modeling and simulation software in the geodynamics and seismology research domains. Scientific computation using software developed and used by the CIG community has helped scientists to understand and quantify the processes that shape the Earth, including convection in the Earth's core and mantle, plate tectonics, the dynamics of plate boundaries with attendant mountain building, volcanism, earthquakes, seismic wave propagation, and the origin and evolution of planetary dynamos. Computational models provide an essential link between observations made on the surface and our attempt to describe the Earth's interior and evolution.

CIG was established in 2005 based on community recognition of the tremendous effort required to develop scientifically sound software, the need to reduce duplication of this effort, and the need to sustain community software beyond an individual "hero" developer. Under this aegis, the CIG research community, with 77 Member Institutions worldwide, gathers to learn from one another through workshops, webinars, and its working groups, to promote ties with computational scientists, establish best practices in coding and training, and support the development and usage of community software. The community works together to build the capacity to support complex, extensible, scalable, interoperable, reliable, and reusable software increasing the return on investment in scientific software development. This increases researchers' ability to explore the critical interdisciplinary challenges of coupled systems using advanced numerical methods, on the large-scale parallel hardware available today. CIG's approach to writing and disseminating scientific software has been described as indispensable to geophysics research (Lay et al., 2012; McGuire et al., 2017; Constable et al., 2016; Williams, 2010).

CIG's repository hosts 32 software packages (Table 1) (*Computational Infrastructure for Geodynamics*), more than half of which are actively used, as evidenced by recently published papers or development activity, by the community for research and education in computational seismology, mantle convection, magma dynamics, short-term and long-term lithospheric deformation, geodynamo, and ice sheet modeling for Earth and beyond. Codes within the repository must meet CIG minimum best practices. Codes must be licensed, open source, use version control, have a portable build system, include tests to verify that it runs properly, have both developer and user documentation, and provide a citable publication. CIG works with developers to bring their codes to these standards and encourages all developers to work toward its target best practices (see https://geodynamics.org/cig/dev/best-practices/).

Accepted software packages are maintained within a community github site and are assigned one of the following support categories:

Developed. Features are actively added by CIG [D_CIG] or by community contributors [D_CONTRIB] to extend or improve capability or performance.

Supported. Actively supported, maintained, and upgraded by CIG [S_CIG] or by community contributors [S_CONTRIB].

Archived. No development activity; not supported. No commitment to updates. May undergo occasional bug fixes [A].

Developed Codes have been validated, passed benchmarks established by the appropriate community, and are leading edge codes in geodynamics. Developed codes may either be donated or developed by CIG staff or the community. These codes are under active development or enhancements and often are actively supported by CIG through maintenance, technical assistance, training, documentation, and testing. These codes are the most frequently downloaded and used.

Supported Codes are mature codes that meet community standards but are no longer undergoing active development. Codes have been benchmarked and documented with examples and references such that they remain useful research tools. Supported codes include codes donated to CIG from members of our community. Minor changes such as bug fixes and binary upgrades are supported. These codes are actively used but may not be as widely used as developed codes.

Archived Codes are legacy codes that have no active development and receive little or no support. Bug reports can be submitted but resources are not readily available to implement fixes.

3. The Approach

To understand and identify prevalent trends and to identify barriers and unanswered questions in how scientific software is developed, maintained, and cited, we examine the practices of the CIG geodynamics community.

We studied this through four separate activities:

1. *Semistructured interviews* included 12 researchers at different career stages from graduate students, postdoctoral scholars, research scientists, and tenure track faculty. Interviewees held positions in university, U.S. government agencies and U.S. national labs, or other research organizations, and self-

identified as computationally skilled domain scientists, computational scientists, and/or software engineers.

2. *Focus groups* held during a software-coding event included 13 predominantly early career researchers (graduate students, postdoctoral scholars and Assistant Professors) from both U.S. and international universities. This group of domain scientists included novice to experienced coders who through their own research interests were contributing code to a single community software project, ASPECT (Bangerth, Dannberg, et al., 2016; Computational Infrastructure for Geodynamics, 2017a, 2017b; Kronbichler et al., 2012).
3. A *survey* distributed to the email lists for communities in the solid earth geosciences—CIG, IRIS, UNAVCO, EarthScope, Geodynamic Processes at Rifting and Subducting Margins (GEOPrisms), and Consortium for Materials Properties Research in Earth Sciences (COMPRES). These communities together have a large constituency who use computational modeling during the course of their research in solid earth geophysics. Eighty-six individuals completed the survey, with the majority being university faculty and early career scientists. The survey and the aggregated results are described in more details in the supporting information (Text S1).
4. *Content analysis* of articles from the 2010–2015 CIG publication list. This list contains 303 publications that the authors themselves identified as using CIG community software, as well as publications identified through a literature search for software names and mentions of CIG. The publication text was mined for software name, citations, additional attributions, website or URL, software version number, and software configuration details (Text S2). Data were analyzed manually in Excel v14.7.2.

All data used in this study is available via Zenodo (Hwang, 2016).

4. Findings

4.1. Learning to Write Scientific Software

Scientific inquiry often leads researchers to questions not yet answered. As these questions push the frontiers, this requires the development of new techniques and tools including software. Today's students have grown up using technology—it enables our everyday lives. However, the acquisition of coding as a skill has lagged (Wilson, 2006). Learning to code and acquiring the skills to write good code are seldom learned through formal instruction in the geosciences. While a large percentage of PhD students in the geosciences take courses on quantitative and computational methods (C. Wilson, 2014) which often mixes in various levels of code learning, these courses do not focus on learning to code and its best practices.

Software development experience among the interviewees reflects these experiences. The majority of participants had a predilection for coding at a young age, but the group as a whole largely learned informally. While they were formally trained in the sciences, they received only informal training in software development. Mentorship (apprenticeship relationships) and peer learning were key in their professional development. Validation and learning from a supervisor or ties developed outside of their institution were significant to advancing their skills. Very few had software-related collaborators at their home institutions. Mentors and in-person interactions through conferences and workshops to build professional and peer networks were highly valued.

The lack of formal software training is typical across the sciences and engineering (Basili et al., 2008; Hannay et al., 2009; Merali, 2010; Prabhu et al., 2011). The National Science Foundation has recognized the need to improve software in research, including training students to write good scientific software (National Science Foundation, 2012). Providing training specifically on scientific software development would decrease the need to teach these skills in domain classes and would promote consistent practices and standards and better use of software tools, all contributing to better scientific software. Inserting research software engineers or trained computer scientists (Basili et al., 2008; Merali, 2010; Hettrick, 2015) into research groups is one way to improve code quality and could also provide mentors for domain scientists interested in developing scientific software.

4.2. Good Code and Coder Conduct

Characteristics of good code as described by interviewees are consistent with best practices recently formally adopted by the CIG community (https://geodynamics.org/cig/dev/best-practices/). These minimum, standard, and target best practices cover version control, coding, portability of configuration and build, testing, documentation, and user workflow in line with current thoughts on best practices for scientific computing (Wilson et al., 2014). While quality is largely a matter of individual judgment, several general intertwined attributes emerged from the interviews. Note that the interviewees were all part of the CIG communities; hence, were predisposed to discussing in terms of code sharing in an open source environment.

Good code is (1) reusable by others, (2) trusted through the inclusion of test cases and as well as its reuse, (3) not overly complex—the underlying ideas are clear, (4) reusable accommodating flexibility of use, (5) readable, understandable to others, (6) consistent in style and syntax, and (7) well documented.

Good coders as described by interviewees were not necessarily defined by their technical abilities, that is, the ability to follow the above practices, but mainly on their behavioral characteristics. While a good coder is described as someone that applies the appropriate technology to the problem and is language agnostic, how the coder conducts themselves within the community defines the majority of traits discussed by interviewees.

A good coder is (1) open to critique, compromise, and direction, allowing for alternative approaches with broader benefit; (2) reasonable in their expectations and patient when coding within a community since software development progresses at uneven tempos; (3) reliable and can focus in the short and long term on project development and cultivation of the community; and (4) a team player who is willing to work on what the community deems important, including the hard, tedious, or fun parts of the project.

4.3. Coding and the Reward System

The academic reward system is traditionally based in part on publication metrics and research impact as demonstrated by quantity and quality of publications in peer-reviewed journals and the impact of those papers. And software? Writing software is not viewed as science among our interviewees even though the majority interviewed were scientists who themselves write software or use specialized software written by their professional colleagues. Rather, software is described as enabling a research program; the time and effort required to create it is considered an obstacle to publishing scientific results. That is, in comparison to peers who did not develop software, interviewees who did spend a significant amount of time developing software perceived themselves as less productive as measured by peer-reviewed publications. Most interviewees with employment security felt lucky that software development had led to academic success. However, attitudes were significantly different among those employed as a domain scientist than those as a computational scientist. Domain scientists viewed effort spent on software development as an obstacle that was overcome “in spite of” apparent loss of productivity whereas those employed as computational scientists viewed it is part of their professional reward system that included contribution to scientific software projects. Within the group of nonacademic interviewees, some hold established positions in which software and software products perhaps can be considered a public good. These participants report that the merit was not judged on scholarly record but evaluated on whether their contributions meshed with their position description.

4.4. Roles in Developing Software

A related issue identified by some interviewees is that the role of *author* of software can encompass a vast array of contributions and is not as well defined as authorship of scientific publications. While tools are emerging to assess both the research effort and the impact of work that is not disseminated through traditional scholarly publications, descriptive attribution for contributors to software lags. For example, Project CRediT (http://casrai.org/credit) recognizes that diverse roles contribute to published research in the sciences and establishes a taxonomy of roles. At a high level, one of these roles is *Software,* defined as: “*Programming, software development; designing computer programs; implementation of the computer code and supporting algorithms; testing of existing code components.*” (http://dictionary.casrai.org/Contributor_Roles/Software).

The taxonomy of *Software* contributions is an active area of exploration for groups concerned with the related topics of sustainability, reproducibility, reusability and discoverability. Organizations such as WSSSPE (http://wssspe.researchcomputing.org.uk/), FORCE11 (https://www.force11.org/), and the codemeta project (http://codemeta.github.io/) have interests in further defining roles in software development as part of assigning recognition for research. The developing concept of altmetrics (e.g. https://www.altmetric.com/, depsy.org and https://impactstory.org/) allows the impact of individuals’ work, whether exhibited in traditional or new media formats, to be captured from online social interaction data e.g. Twitter followers, Mendeley readership. While these projects help to recognize software contributions in new ways, contributions to a project remain subjective and hard to quantify.

Within its development community, a software project does not thrive without *leadership* and a community that engages in roles as *maintainers*, *contributors*, and *users* (Bangerth & Heister, 2013). The CIG community's projects' structure and size are characteristic of most open source software projects in the geosciences. Development communities range from small (~3) to very large (~50) with substantially larger user bases (greater than hundreds to thousands).

CIG software communities require *leadership* from within the scientific domain, to ensure that the software development meets a clear scientific need. Leaders establish the intellectual framework for a software project, either through formal processes for community input or informally through charismatic influence. In the CIG community, where many scientists begin by developing their own software, leaders are often the project's original software architect(s). These leaders are highly engaged in the survival and continuity of a project, remaining influential even after their direct involvement in the software development diminishes.

Users are attracted to a software package when they perceive it as usable, applicable, and an improvement over other available tools. They become *contributors*, reporting bugs, submitting patches and implementing new features as the nature of their research problem outpaces the existing development efforts. Becoming a *maintainer* requires additional technical and social skills to support the software infrastructure. These skills include reviewing code, responding to open issues and user requests, and growing and nurturing the user base, eventually developing new maintainers. Members of a software community may fulfill multiple roles, their roles may change over time, and membership in the community can change over time.

In addition to these software development roles largely filled by the geodynamics community, CIG employs a software engineering staff. The organization and its staff make the software available, robust, and usable. In accordance with its best practices, CIG hosts a software repository for version control and issue tracking, maintains continuous integration testing, maintains documentation, and responds to routine help requests and bug fixes.

As practiced by CIG, this system obscures contributions to software development and engineering. Peers easily recognize each other's contribution through interaction in peer networks including software repository activity, and by mining information on organizational or project websites. From monitoring a project's e-mail lists or repositories, a member can learn who are the experts and key contributors. Websites and manuals may indiscriminately list all contributors, a select few, or only the leadership. Regardless, institutional reward systems often neglect these contributions because they are presented in non-traditional forms, that is, outside the peer-reviewed publication system.

4.5. Current Practice in Software Citation

CIG, like many communities, uses existing mechanisms to provide credit for software and is only just moving to persistent identifiers. One of CIG best practices states that developers of hosted codes must specify what to cite and acknowledge. This information is disseminated through user documentation and workshops. Our survey showed that developers ask users to acknowledge scientific software by citing both published peer-reviewed papers and non-peer-reviewed media such as user manuals, technical reports, or software landing page websites. However, software author lists are not comprehensive and are difficult to interpret (Dance, 2012; Hornik et al., 2012; Tscharntke et al., 2007). For example, the authorship list on CIG software manuals is typically alphabetical for large groups with no indication of what features or how an author contributed, while published peer-reviewed papers in geophysics usually follow the first author norms with the major contributors listed first or with authors listed alphabetically.

As one of CIG's best practices, developers are asked to provide at least one citable publication. For just over half of the CIG hosted software packages, this is a peer-reviewed publication for citation. Most often this is a science paper describing the algorithm or the underlying methods the software implements in a section devoted to methods, where a scientific finding is the main focus of the paper. Much less frequently the developers provide a peer-reviewed paper describing the software, the user manual, and/or project website. An additional attribution in the form of acknowledgement to a funding agency or the repository is frequently requested (Figure 1). A significant number of software packages make

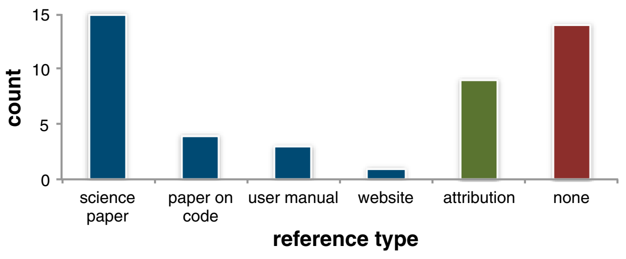

Figure 1. The CIG repository maintains 32 software packages. Developers either ask users to cite a *science publication* that describes the software as part of a scientific application, a *publication on software* that describes the software as the main subject of the paper, the software's *user manual,* and/or the software's *website*. Additional *attributions* are sometimes requested. The group of software packages requesting *none* comprises mainly legacy software, tools, and/or libraries that have no attribution information.

no attribution request. The majority of these are legacy software (Archived) that are no longer in widespread use, while others are variants, tools or libraries.

Given the availability of a citable publication, the next relevant question is: how regularly do users invoke these citations in scientific publications that use the software? To assess this, we carried out an analysis of 303 papers published between 2010–2015 by the CIG community. Publications were either provided by the authors in response to CIG's regular request for publications using CIG community software, or were obtained through a literature search for publications identifying software by name or mentioning CIG. Hence, this represents publications from a highly engaged group of contributors. The papers were predominantly published in journals with impact factors between 2.5–3.5 (Figure 2).

We read these papers for mentions of software by name supplemented by key word search, citations, additional attributions, website or url, version number, and configuration detail. Software mentions were most often found in the "methods" section of publications but were also found in the acknowledgements, appendices, figure captions, addendums, and electronic supplements.

Within this group of publications, 83% of the papers mention at least one CIG-hosted software package. That some publications do not cite any CIG software can be attributed to several factors. Some papers describe the mathematical methods and algorithms used in the software packages rather than discussing or using a particular software package; the list also includes review papers that discuss relevant topics but do not present original research. Neither type of publication cites any specific software. After removing those examples, there still remains a subset of reported papers in which modeling software was clearly used and hence, the software should have been named in the paper.

When a software package was named in the text of the publication, 75% of the papers also provided a citation in the references. This includes software packages that did not formally provide a preferred citation in their documentation or website. Citations show good compliance with the instructions on how to cite provided in the software's user manual with fewer than 10% of the papers not citing any of the requested publications. For those packages providing no formal instructions on how to cite, 70% of these papers provided a citation. This is in good agreement with survey results in which 93% of the respondents responded that they cite software used in course of their research. Several survey respondents also volunteered that they seek to cite as requested by the software developers.

For comparison, Howison and Bullard (2015) examined a randomly selected sample of biology papers to assess the visibility of software in publications. In their study, 65% of the articles mentioned software with a higher likelihood of mention in higher impact journals. Of these 44% provided a citation in the reference list. The comparison between the two studies is not perfect in that the publication list used in our analysis was self-selected by the CIG community, a group with a professional interest in seeing software authorship given credit. We thus would expect, and in fact observe, a higher rate of citation in the CIG publications than in the randomly selected publications reported by Howison and Bullard.

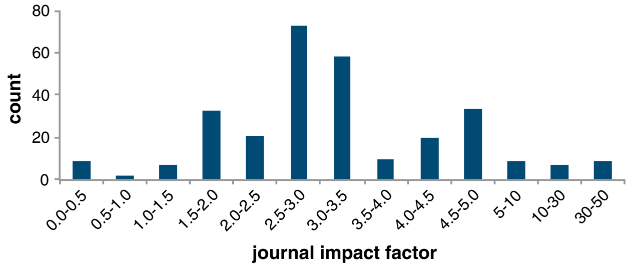

Figure 2. Impact factors for journals in which the CIG community publishes as reported by each journal in 2015.

The CIG community surveyed recognized that software used for research should be cited in the related scholarly outputs but there is disagreement about what software should be cited and how to cite. The group is in clear consensus that full software packages should be cited but there is no consensus about whether and how to cite additional software used in the processing, manipulating, and visualization of data, nor about citing scripts, libraries, and commercial packages. In the case of commercial software, interviewees felt that commercial software developers were compensated for their labor directly via wages or royalties, so additional credit in the form of citation was not necessary.

The publication record supports the general idea that more than just the modeling software packages should be cited, as software was mentioned

500 times by name. What was considered citable varied widely and included commercial and open source software used in constructing models, processing data, analysis, and visualization. Packages named included meshing software such as CUBIT (CUBIT, 2017), LaGriT (LaGRIT, 2017) and Abaqus (Dassault Systemes, 2017); MATLAB (Mathworks, 2017), a high-level language and interactive environment for numerical computation, visualization, and programming; GPlates (GPlates, 2017) for plate tectonic reconstructions, Paraview (Kitware, 2017), a data analysis and visualization package; and SAC, a Seismic Analysis Code (Goldstein et al., 2003; Goldstein & Snoke, 2005).

Only 21% of the analyzed publications contain an URL leading to a software landing page or an organizational website and none included persistent identifiers. Even fewer, 13%, provide sufficient information to identify a version number of the cited software. In addition, authors rarely reported which software features were used and whether the software had been modified in the course of the research. Those surveyed acknowledged version is important because software continues to change over time with the addition of new features and bug fixes.

Of the non-CIG software packages, one stood out – GMT the Generic Mapping Tools (GMT) (Wessel et al., 2013; Wessel & Smith, 1991, 1995, 1998). GMT is a well-established open source collection of tools for map creation and was mentioned in 13% of the papers. The tool has been in use by the geoscience community for more than 20 years and currently is in release version 5. Releases are accompanied by articles in EOS Transactions AGU—the weekly earth science magazine of the American Geophysical Union. The authors' request for citation dates back to 1991. The large number of citations observed in this study can likely be attributed to both its rich tradition of citation in the literature and that the developers provided a citable article.

5. Discussion of Open Issues

The groups here represent a unique part of the geoscience community; they are researchers who both use and develop scientific software, some of whom have found a career path through coding. Hence, they are favorably disposed to receiving and providing credit for software as expressed through their willingness to participate in these surveys and interviews and reporting of publications to CIG. In practice, these users are citing software by adapting established mechanisms for scholarly credit such as citing a peer-reviewed publication, even when these mechanisms do not adequately convey the status, capabilities, or authorship of the software used. Requesting citations in most cases satisfies the traditional merit systems, but does not support emerging demands to support software reuse and scientific reproducibility.

In addition to the active researchers and software developers we surveyed, those interested in accurate attribution of credit for scientific software also may include software developers who have left the research environment, project managers, funding agencies, publishers and scientific societies, and library scientists. Each of these has a different perspective on the value of software, which was not investigated here but should be considered for any attribution system, for instance funding agencies are interested in assessing outcomes and impacts, while library scientists are also concerned about discovery, access and preservation.

Additionally, organizations that maintain or support software development, software repositories, or data centers would be aided in assessing the value and impact of their investments by a clearer identification of their role. Some metadata specifications include such roles (*see http://codemeta.github.io/)* but there is no established practice in the literature on how to acknowledge organizations. This question is complicated by the different roles played by organizations, from codeveloper to curator of software. CIG, for example, requests acknowledgement for its role but in this study, only 19% of the papers complied with this request.

Assessing contributions to software, however, is not the same as measuring the quality or impact of the software or the contribution. Presently, there are no standardized mechanisms for evaluating code. Easily quantifiable metrics such as lines of codes or number of commits (number of revisions to a file(s) in a software repository such as Github) can easily be manipulated and say nothing about whether the software is correct. Refactoring code (restructuring the software without changing its external behavior) can, in the aggregate lead to fewer lines of code. Essential changes to a code's core algorithm have a different impact than, for example, modifications to input/output formats. Comments in software, while not strictly "code" can increase the number of lines and increase the usability, or not, of the code; similarly with readability fixes. Changes can be made via a series of single commits or multiple changes can be aggregated into one commit. Most metric systems are not beyond gaming by an astute community (Abbott et al., 2010; Nightingale & Marshall, 2011; Sahel, 2011). Assessing contribution, assigning credit, and documenting provenance of software remains a challenge. For example, the contribution of the software engineer who tests the software to ensure that updates do not break it but never touches the code itself remains unseen in all current measures.

Without prompting, survey respondents overwhelmingly expressed a desire for two improvements in software citation. The first was a DOI or other persistent identifier for software which would enable recognition of authorship and code contribution. The second desire was a generalized call for some standardized format, guidelines, or instructions for citing software and updating that citation to reflect different versions of the software. These needs have been recognized across a broad range of research domains attracting the interest of funders and agencies e.g. the National Science Foundation, National Institute of Health, and the Alfred P. Sloan Foundation.

Activities and projects of this international community include:

1. Force11 Software Citation Working group published draft Software Citation Principles for broad adoption among the various stakeholder communities. (Smith et al., 2016).
2. WSSSPE has run a series of workshop providing a forum for problems relating to the role of software in research including credit (http://wssspe.researchcomputing.org.uk/) (Katz et al., 2014; Katz, Choi, Niemeyer, et al., 2016; Katz, Choi, Wilkins-Diehr, et al., 2016).
3. The Software Sustainability Institute (SSI) provides guidelines on how to cite software http://software.ac.uk/so-exactly-what-software-did-you-use.
4. Zenodo (https://zenodo.org/) provides an online repository for software and its metadata and can provide DOIs.
5. OntoSoft (http://www.ontosoft.org/) facilitates capture of metadata and software publication providing training that supports software citation and credit (Gil et al., 2015).

One interesting and unexpected attitude about software emerged from this study. An apparent tension exists between the communities that are interested in the reuse of software and those that develop software. According to interviewees, “good code” is reusable by others, however, “science wants to do something different every time”. This implies that the goal of software is replicability or repetition, whereas, the goal of science is discovery and novelty. However, software is frequently reused in research. Numerical libraries such as PETSc (Balay et al., 2016a, 2016b, 1997), deal.II (Bangerth et al., 2007; Bangerth, Davydov, et al., 2016) are under active development and exists as stand alone research projects which are actively used by research projects worldwide. When a new software or computational method is developed, a suite of data or use cases can be applied that expands our knowledge. Whether studies can proceed without any alteration of the software, speaks to the consistency of the data, the flexibility of the software (best practice), and broad applicability of the question being asked. The reuse of scientific software is key to advances in science dependent on computation, reduces duplication of effort, and improves its quality by utilizing well-tested components.

CIG currently does not assign identifiers to their software packages. However the community is moving forward in piloting DOIs using Zenodo (www.Zenodo.org). Zenodo provides integration to github and supports communities, and using containers to preserve workflow. The creation of the CIG community creates visibility for those interested in obtaining DOIs for their research objects and discoverability for others. This is part of a larger CIG effort to provide a simple mechanism for software attribution for stakeholders.

6. Conclusions

Today, software is necessary for scientific advancement. It increases capacity to process and analyze data, making science more efficient, enabling analysis and modeling of data and processes on an unprecedented scale to test hypotheses, make predictions, and advance knowledge. However, to effectively use these new tools, scientists must develop computational and coding skills, and knowledge to expand the pool of answerable questions. From this study, it is clear that the human resources of ingenuity and skill applied to scientific computation are currently being recognized, applied, rewarded, and documented only on an *ad hoc* basis. Training across the sciences is being supplemented through organizations such as *Software and Data Carpentry* (G. Wilson, 2014), *Khan Academy*, and *Google Summer of Code*. These appeal to those predisposed to coding but do not make such training widely accessible to all who could benefit. Incorporating formal, practical, coding courses into curricula remains challenging.

Receiving credit for code is only one of many issues related to software sustainability and reproducibility. Software sustainability requires the training and retention of developers beyond those who “got lucky” to have support from mentors and the creation of institutions that value their skills. Creating a recognition system to get credit for code will increase visibility for various roles and make more transparent the effort good code requires. Toward that goal, is a movement to recognize Research Software Engineers as promoted by UKRSE (http://www.rse.ac.uk/events.html) job titles for those who work in research but write code and not papers and may work within a domain department or a research support center. Developing a “software prize” has been discussed as recognition that would not depend on publication (Bangerth, Dannberg, et al., 2016; Bangerth, Davydov, et al., 2016). Lastly, reproducibility, a hallmark of good science, has been under increasing scrutiny (McNutt, 2014). For numerical models, if the software becomes lost to the community, research results cannot be verified. Citing software is the first step toward addressing these issues.

Our results indicate that awareness of researchers for the need to cite is high and that more importantly, researchers seek to cite. When given information on how to cite, many authors use the requested citation provided by developers. However, developers need to be educated to include this information in their documentation. Users as well need to be educated on what to cite while recognizing that this is an area of active discussion and the community norms will continue to evolve. CIG is moving toward issuing unique identifiers for its repository, creating tools for easy attribution, and educating the community on recommended practices for citation.

Acknowledgments
We thank all those who participated in interviews, focus groups and the survey for this study. We thank Bakinam T. Essawy, Yolanda Gil, and Allen Pope for their reviews. This project is supported by the National Science Foundation award number SMA-1448633. The Computational Infrastructure for Geodynamics is supported by the National Science Foundation award NSF-0949446. The data for this study is freely available through the Zenodo repository https://doi.org/10.5281/zenodo.158485.

References

Abbott, A., Cyranosk, D., Jones, N., Maher, B., Schiermeier, Q., & Van Noorden, R. (2010). Metrics: Do metrics matter? *Nature*, *465*, 860–862. https://doi.org/10.1038/465860a

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., … Zhang, H. (2016a). {PETSc}{W} eb page.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., … Zhang, H. (2016b). PETSc users manual revision 3.7. Argonne, IL: Computer Science Division, Argonne National Laboratory.

Balay, S., Gropp, W. D., McInnes, L. C., & Smith, B. F. (1997). Efficient management of parallelism in object-oriented numerical software libraries. In *Modern software tools for scientific computing*, (pp. 163–202). Boston: Birkhäuser.

Bangerth, W., Dannberg, J., Gassmoeller R., & Heister, T. (2016). ASPECT: Advanced Solver for Problems in Earth’s Convection v1.4.0, *Computational Infrastructure for Geodynamics*.

Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., … Wells, D. (2016). The deal. II library, version 8.4. *Journal of Numerical Mathematics*, *24*(3), 135–141.

Bangerth, W., Hartmann, E. R., & Kanschat, G. (2007). deal. II—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, *33*(4), 24.

Bangerth, W., & Heister, T. (2013). What makes computational open source software libraries successful? *Computational Science & Discovery*, *6*, 015010.

Basili, V., Carver, J., Cruzes, D., Hochstein, L., Hollingsworth, J., Shull, F., & Zelkowitz, M. (2008). Understanding the High-Performance-Computing Community: A Software Engineer’s Perspective. *IEEE Software*, *25*(4), 29–36. https://doi.org/10.1109/ms.2008.103

Chue Hong, N. (2014) In which journals should I publish my software? Software Sustainability Institute, *Software.ac.uk*. Retrieve from http://www.software.ac.uk/resources/guides/which-journals-should-i-publish-my-software, (Accessed 13 July 2016)

Computational Infrastructure for Geodynamics (2017a). List of Software, *Computational Infrastructure for Geodynamics: Software*. Retrieve from https://geodynamics.org/cig/software/, (Accessed 20 June 2017)

Computational Infrastructure for Geodynamics (2017b). ASPECT. Retrieve from https://geodynamics.org/cig/software/aspect/, (Accessed on 20 January 2017)

Constable, C. G., Masters, T. G., Buffett, B., Day, J. M. D., Hirschmann, M., Karato, S.-I., … Mao, W. (2016). Cooperative Studies of the Earth’s Deep Interior: Understanding the origin and evolution of our planet through interdisciplinary research. Retrieved from csedi.org/2016_Report

CUBIT (2017). Retrieve from https://cubit.sandia.gov/, (Accessed 20 January 2017)

Dance, A. (2012). Authorship: Who’s on first? *Nature*, *489*(7417), 591–593. https://doi.org/10.1038/nj7417-591a

Dassault Systemes (2017). Abaqus. Retrieve from http://www.3ds.com/products-services/simulia/products/abaqus/, (Accessed on January 2017)

Gil, Y., Ratnakar, V., & Garijo, D. (2015). OntoSoft, *Proceedings of the Knowledge Capture Conference on ZZZ - K-CAP 2015*. https://doi.org/10.1145/2815833.2816955.

Goldstein, P., Dodge, D., Firpo, M., & Minner, L. (2003). SAC2000: Signal processing and analysis tools for seismologists and engineers. In W. H. K. Lee, et al. (Eds.), *Invited contribution to “The IASPEI International Handbook of Earthquake and Engineering Seismology”* (pp. 1613–1620). London: Academic Press.

Goldstein, P., & Snoke, A., (2005). “SAC Availability for the IRIS Community”, Incorporated Institutions for Seismology Data Management Center Electronic Newsletter.

GPlates (2017). Retrieve from https://www.gplates.org/, (Accessed 20 January 2017)

Hannay, J., MacLeod, C., Singer, J., Langtangen, H., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software?, *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. https://doi.org/10.1109/secse.2009.5069155

Hettrick, S.. (2015). Why we need to create careers for research software engineers - Scientific Computing World, Retrieve from Scientific-computing.com, http://www.scientific-computing.com/news/news_story.php?news_id=2737, (Accessed 13 July 2016)

Hornik, K., Murdoch, D., & Zeileis, A. (2012). Who Did What? The Roles of R Package Authors and How to Refer to Them. *The R Journal*, *4*(1), 64–69.

Howison, J., & Bullard, J. (2015). Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology*, *67*(9). https://doi.org/10.1002/asi.23538

Hwang, L. (2016). CIG Publications 2010–2015 Dataset. *Zenodo*. https://doi.org/10.5281/zenodo.158485

Katz, D., Choi, S.-C. T., Wilkins-Diehr, N., Chue Hong, N., Venters, C. C., Howison, J., … Littauer, R. (2016). Report on the Second Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2). *Journal of Open Research Software*, *4*. https://doi.org/10.5334/jors.85

Katz, D. S., Choi, S.-C. T., Lapp, H., Maheshwari, K., Löffler, F., Turk, M., … Venters, C. (2014). Summary of the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1). *Journal of Open Research Software*, *2*(1), e6. https://doi.org/10.5334/jors.an

Katz, D. S., Choi, S.-C. T., Niemeyer, K. E., Hetherington, J., Löffler, F., Gunter, D., … de Val-Borro, M. (2016). Report on the Third Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE3), abs/1602.02296. *Software Engineering*. arXiv:1602.02296

Kelly, D., Hook, D., & Sanders, R. (2009). Five Recommended Practices for Computational Scientists Who Write Software. *Computing in Science & Engineering*, *11*(5), 48–53. https://doi.org/10.1109/mcse.2009.139

King, S. D., Raefsky, A., & Hager, B. H. (1990). Conman: vectorizing a finite element code for incompressible two-dimensional convection in the Earth's mantle. *Physics of the Earth and Planetary Interiors*, *59*(3), 195–207. https://doi.org/10.1016/0031-9201(90)90225-M

Kitware (2017). Retrieve from http://www.paraview.org/, (Accessed 20 January 2017)

Kronbichler, M., Heister, T., & Bangerth, W. (2012). High accuracy mantle convection and simulation through modern numerical methods. *Geophysical Journal International*, *191*(1), 12–29. https://doi.org/10.1111/j.1365-246X.2012.05609.x

LaGRIT (2017). Retrieve from http://lagrit.lanl.gov, (Accessed 20 January 2017)

Lave, J., & Wenger, E. (1991). *Situated learning*. Cambridge, England: Cambridge Universitys Press.

Lay, T., Bender, M. L., Carbotte, S., Farley, K. A., Larson, K. M., & Lyons, T., … Zhang, D. (2012). *New Research Opportunities in the Earth Sciences at the National Science Foundation* (117 pp.). Washington, DC: National Research Council, The National Academies Press.

Lehnert, K., Su, Y., Langmuir, C., Sarbas, B., & Nohl, U. (2000). A global geochemical database structure for rocks. *Geochemistry, Geophysics, Geosystems 1*, 1012. https://doi.org/10.1029/1999GC000026

Mathworks. (2017). MATLAB. Retrieve from https://www.mathworks.com/products/matlab.html, (Accessed 20 January 2017)

McGuire, J. J., et al. (2017). The SZ4D Initiative: Understanding the Processes that Underlie Subduction Zone Hazards in 4D. Vision Document Submitted to the National Science Foundation. The IRIS Consortium, 63 pp.

McNutt, M. (2014). Reproducibility. *Science*, *343*(6168), 229. https://doi.org/10.1126/science.1250475

Merali, Z. (2010). Computational science: ...Error… why scientific programming does not compute. *Nature*, *467*(7317), 775–777. https://doi.org/10.1038/467775a

National Science Foundation (2012). A vision and strategy for software for science, engineering, and education: Cyberinfrastructure framework for the 21st century, NSF Document Number nsf12113, September 21, 2012.

Nightingale, J. M., & Marshall, G. (2011). Citation analysis as a measure of article quality, journal influence and individual researcher performance. *Radiography*, *18*(2), 60–67. https://doi.org/10.1016/j.radi.2011.10.044

Oberkampf, W., & Roy, C. (2010). *Verification and validation in scientific computing*. New York: Cambridge University Press.

Prabhu, P. Kim, H., Oh, T., Jablin, T. B., Johnson, N. P., Zoufaly, M., Raman, A., Beard, S. (2011). A survey of the practice of computational science, *State of the Practice Reports on - SC '11*. https://doi.org/10.1145/2063348.2063374

Reed, D. A., Bajcsy, R., Fernandez, M. A., Griffiths, J.-M., Mott, R. D., Dongarra, J., … Polnick, T. L. (2005). *Computational science: Ensuring America's competitiveness*. Arlington, VA: President's Information Technology Advisory Committee.

Romanowicz, B., Neuhauser, D., Bogaert, B., & Oppenheimer, D. (1994). Accessing northern California earthquake data via Internet. *Eos Transactions of the American Geophysical Union*, *75*(23), 257–260. https://doi.org/10.1029/94EO00934

Sahel, J.-A. (2011). Quality Versus Quantity: Assessing Individual Research Performance. *Science Translational Medicine*, *3*. 84cm13.

Smith, A. M., Katz, D. S., & Niemeyer, K. E., & FORCE11 Software Citation Working Group (2016). Software Citation Principles. *PeerJ Computer Science*, *2*, e86. https://doi.org/10.7717/peerj-cs.86

Trabant, C., Hutko, A. R., Bahavar, M., Karstens, R., Ahern, T., & Aster, R. (2012). Data Products at the IRIS DMC: Stepping Stones for Research and Other Applications. *Seismological Research Letters*, *83*(5), 846–854. https://doi.org/10.1785/0220120032

Tscharntke, T., Hochberg, M., Rand, T., Resh, V., & Krauss, J. (2007). Author Sequence and Credit for Contributions in Multiauthored Publications. *PLoS Biology*, *5*(1), e18. https://doi.org/10.1371/journal.pbio.0050018

Wessel, P., & Smith, W. H. F. (1991). Free software helps map and display data. *EOS Transactions of the American Geophysical Union*, *72*, 441.

Wessel, P., & Smith, W. H. F. (1995). New version of the Generic Mapping Tools released. *EOS Transactions of the American Geophysical Union*, *76*, 329.

Wessel, P., & Smith, W. H. F. (1998). New, improved version of the Generic Mapping Tools released. *EOS Transactions of the American Geophysical Union*, *79*, 579.

Wessel, P., Smith, W. H. F., Scharroo, R., Luis, J. F., & Wobbe, F. (2013). Generic Mapping Tools: Improved version released. *EOS Transactions of the American Geophysical Union*, *94*, 409–410.

Williams, Q. (Ed.) (2010). Understanding the Building Blocks of the Planet: The Materials Science of Earth Processes. Report to the National Science Foundation. COMPRES Consortium (68 pp.).

Wilson, C. (2014). *Status of Recent Geoscience Graduates*. Alexandria, VA: American Geosciences Institute. 0-922152-99-3.

Wilson, G. (2006). Software carpentry: Getting scientists to write better code by making them more productive. *Computing in Science & Engineering*, *8*(6), 66–69.

Wilson, G. (2014). Software Carpentry: lessons learned, *F1000Research*. https://doi.org/10.12688/f1000research.3-62.v1

Wilson, G., Aruliah, D. A., Titus Brown, C., Chue Hong, N. P., Davis, M., Guy, R. T., … Wilson, P. (2014). Best Practices for Scientific Computing. *PLoS Biology*, *12*(1), e1001745. https://doi.org/10.1371/journal.pbio.1001745