Scientific software production and collaboration

Author 1
Affiliation
Affiliation
author@a.com

Author 2
Affiliation
Affiliation
author2@b.com

ABSTRACT
Software plays an increasingly critical role in science, including data analysis, simulations, and managing workflows. Unlike other technologies supporting science, software can be copied and distributed at essentially no cost, potentially opening the door to unprecedented levels of sharing and collaborative innovation. Yet we do not have a clear picture of how software development for science fits into the day-to-day practice of science, or how well the methods and incentives of its production facilitate realization of this potential. We report the results of a multiple-case study of software development in three fields: high energy physics, structural biology, and microbiology. In each case, we identify a typical publication, and use qualitative methods to explore the production of the software used in the science represented by the publication. We identify several different kinds of production systems, characterized primarily by differences in incentive structures. We identify ways in which incentives are matched and mismatched with the needs of the science fields, especially with respect to collaboration.

ACM Classification Keywords
H.5.3 Group and Organization Interfaces: Computer-supported cooperative work

INTRODUCTION
Software plays an increasingly critical role in science, including data analysis, simulations, and managing workflows. Unlike other technologies supporting science, software can be copied and distributed at essentially no cost, potentially opening the door to unprecedented levels of sharing and collaborative innovation.

While science is a collaborative field, both in terms of specific projects and indirectly over time as research projects build on each other, it is not selfless. Scientific software work takes place in the context of competition amongst scientists for recognition and attention that focuses on publications and citations. This environment makes the incentives and rewards for software work quite different than that found in other domains.

We do not have a clear picture of how software development for science fits into the day-to-day practice of science, or how well the methods and incentives of its production facilitate realization of the potential of software in science.

BACKGROUND
Software work in science has received limited research attention, despite its growing importance. What work there is has been focused in two areas: Collaboratories and Software Engineering.

The literature on collaboratories has focused on software as an infrastructural support for collaboratory science, such as in the UARC scientific collaboration Upper Atmospheric Research Collaboratory [3, 9]. A common finding is a division of labor between scientist and computer scientists or software developers creating a tension between research goals and software development goals [8, 10, 2, 5].

In Software Engineering "a chasm opened between the scientific-computing community and the software engineering community ... the bulk of the software engineering community's research is on anything but scientific-application software" [6]. When SE has approached scientific software it has focused on the idea that SE methodologies can help assess, test and improve software correctness [4, 1]. The recommendations in this work often exhort scientists to adopt imported practices, yet they do without understanding what underlies and thus causes the situation they seek to fix.

What is needed, as we see it, is to understand scientific software as an independent production system, by which we mean examining the incentives underlying its production and considering their impact on the practices and the software produced. Knowing what drives and constrains current practices forms the basis for developing interventions which can improve the situation.

There are two examples of work towards this end. [11] is a survey of scientific software that specifically considers the organization of its production, such as whether it was produced by commercial interests or open source projects, albeit as one source of correctness risk. Segal [12, 13, 14] approaches scientific computing through the lens of end-user software development, a branch of SE research that recognizes that the practice of software development regu-

Submitted for review to CSCW 2011.

larly occurs outside the traditional role of professional software developers. [13] describes field studies of two modes of software development in science. She identifies two situations: "when the software is intended for use either by the developer herself/himself or by closely co-located colleagues, for example, people working in the same laboratory" which she contrasts with software developed "within a closely co-located group but intended for the wider scientific community of which the developers form a part." While these distinctions in practice and intended use are important we believe that they can be improved through a focus on the incentives faced by scientists producing software and that this approach is most likely to yield actionable knowledge.

THREE CASES

We therefore began this research with a set of sensitizing concepts: incentives, collaborative practices and software correctness in the scientific context. As we conducted these cases we refined our questions to four that form a convenient way to present our results.

1. What software is involved in this scientific field?
2. Who created or maintains this software?
3. What incentives drive that creation or maintenance work?
4. Why is the software trusted?

Since each scientific field is very large we needed to ground our cases. This we did by identifying specific published papers which we refer to as *focal* papers. Since a published paper is the primary scientific unit of work and contribution it is an appropriate route into these fields. While we confirmed with our interviewees that our focal papers represented a "typical" workflow in their work and field, we present these as individual cases of software heavy science in each field. This choice, of course, allows us to provide rich detail at the risk of representativeness; we judged this tradeoff to be appropriate at this stage of inquiry.

For each specific publication we interviewed the authors most associated with the software work of the paper. These interviews provide direct answers to the first question, allowing us to identify the software involved. This includes software written or directly used by the authors in the course of the research leading to the publication of the focal paper. Yet the involvement of software does not stop there. We probed beyond the surface, seeking additional software relevant to these publications. As we will see this revealed software involved in, for example, creating datasets used by the authors or in performing analyses or simulations needed for the work but performed by others.

For each of the software artifacts identified we pushed outwards. We first sought to understand how the authors came to have the software they used, then we sought to understand the origins of all the software involved, arranging interviews with some package authors. These interviews were complimented by more general interviews with scientists and software practitioners in the field (some conducted prior to identifying our specific paper, some afterwards). In this way we build a picture of the scientific software ecosystem involved in these specific pieces of science.

Following the three cases we provide an extended Discussion which builds a taxonomy of software production systems in terms of their incentives, practices and logics of correctness.

CASE 1: LARGE PHYSICS COLLABORATIONS

The first case we present is science in large common-goal collaborations. We have conducted interviews with participants in a number of collaborations in physics, including STAR, ICECube, CMS, ATLAS and LIGO. These collaborations are large, comprised of hundreds of scientists all focused on a relatively narrow, shared goal [7]. The collaborations are literally centered on large scientific instruments, from particle accelerators to cubic kilometer instrumented blocks of ice in the Antarctic which generate and measure physical conditions of interest. The method uses simulations of theoretically understood processes which are subtracted from the massive collected data to isolate and characterize an extremely small number of interesting events. These collaborations typically have substantial central funding (often through multiple, overlapping grants over long time-frames) complemented by a range of smaller grants to member labs focused on particular aspects of the projects.

Focal publication Our focal paper for this case was produced by the STAR high-energy physics collaboration. It characterizes the Upsilon particle, including statistical bounds for its energy range. This paper was identified through an interview with the software coordinator for the STAR collaboration.

The first thing that strikes one about the paper is that it includes the full list of collaboration members (including central IT staff) as authors (taking up the first two pages) and does not indicate who was particularly responsible for this analysis. This is normal practice in these collaborations. The software coordinator was able to share an internal tech note which describes the analysis in detail intended to be sufficient for replication inside the collaboration. This lists six scientists as "the primary authors of this paper"; the software coordinator identified the two listed first as primarily responsible for the software portions of the paper and we interviewed them. The tech note provides details of the software most closely associated with the analysis; this formed the basis for our interview with the authors.

Identifying software involved

The instrument run creates raw data, which is "pre-processed" from electrical signals into initial descriptions of "events" in the detector. The analysis begins with this data and works through 25 scripts known by the authors as "the analysis macros." The first set of scripts, called "Makers," accesses a particular subset of a particular run of "pre-processsed data" together with a specific version of a software library called ROOT4STAR, ensuring that it is synchronized with the dataset. The "Data" set of scripts perform the physics analysis itself. The remaining scripts assess "systematic uncertainties" in the analysis, using a specific simulation practice called "embedding." Producing these simulations involve more software.

Thus the full workflow for the paper draws on four kinds of software: analysis macros, the ROOT4STAR library, the software involved in data production and the software involved in simulation production. For each of these we now consider our three questions, addressing its creation, incentives and its logic of correctness.

Analysis scripts The authors of the analysis scripts were two younger scientists, one a post-doc and one a senior PhD student. They worked in different labs at different institutions. Each had some background in Computer Science consisting of 2 or 3 years in an undergraduate CS major before focusing on Physics through graduate school.

One scientist was responsible for the Makers and the Data scripts and the other for the Embedding scripts, but they described being in regular touch through email and phone. Each scientist worked primarily on his or her laptop, prototyping their scripts on small sub-sections of the data, before running them in user accounts on "STAR Cluster machines." They shared their code through directories on this system and did not use source control, considering it unnecessary for small groups during the preparation of the analysis. They did not write tests or use any particular software development methodology.

Their work unfolded in the context of weekly meetings of their physics analysis group. The analysis working group includes multiple labs and people experienced with the instrument and similar analyses. The group assess the credibility of the results in the context of their knowledge of both the instrument, results being produced elsewhere in the collaboration and the physics theory which links them. In the words of the authors, "If you are doing something that looks really strange, or you have a result that doesn't make sense, they can give you a heads up right away." This iterative, enveloping process, plus the fact that they wrote the code themselves, were the main logic of correctness used by the authors.

Once the group has decided to move to publication the collaboration organizes a "godparent" committee to review the paper. At this point the analysis scripts are moved into a shared CVS and tagged; a policy insisted upon by the central software coordinator. The review undertaken by this committee focuses on the physics findings, but also assesses the completeness of the tech note. It includes a scientist member specifically responsible for assessing the software. That person runs portions of the analysis to confirm that they get the same results. In the case of this paper the only issues uncovered in this process was a script missing from the central CVS. This review was mentioned only secondarily as a reason to believe in the integrity of the code; it was primarily a check on reproducibility.

The incentives to write this software were clear and direct: without these scripts there would be no analysis and thus no paper. In the words of a member of the central IT staff of a different collaboration, scripts like these are "all about getting the plots," where the plots refer to the results forming the central findings of the paper.

Our informants believed that their greater than average technology skills did provide some comparative advantage within the collaboration. Nonetheless they were clear that they expected to be judged on physics results obtained, not their software skills. They argued that writing code was something that was intensely important during the early stage of one's career, but that career progression would lead them away from writing software. They speculated that focusing entirely on software contributions not directly linked to specific results would lead to a move out of the main career track; they were unsure how that would work. They indicated that their career progression would rely on their reputation within the collaboration, especially on letters written by their PIs. They expected these letters to mention their software work only in the context of their scientific competence.

Our informants told us that they "of course" expect their code to be open to review and for the use of anyone interested, within the collaboration. The issue of sharing outside the collaboration did not arise, in their minds, since the code was so specialized. They did not expect to support others using their software since they believed that it was too specialized and in any case the code's availability and documentation through the Tech Note ought to be sufficient. From time to time they had sought code from others in the collaboration, mainly as a demonstration of portions of the analysis or plot generation, and had found everyone basically forthcoming, although some were quicker and more helpful than others.

ROOT4STAR Library The ROOT4STAR library is fundamental to the STAR collaboration. It is a specialization of a library called ROOT that is created outside of STAR and is widely used in the physics community. ROOT4STAR deals with data input/output issues, optimized implementations of algorithms needed through the collaboration and plotting tools.

ROOT4STAR was constructed with two main contributions. The first are from "5-7" software professionals (often with physics backgrounds) working in a "central IT group." The second are collaboration members, often students or post-docs, assigned from member labs to provide "service work." This "service work" is a tradition in physics collaborations. Originally it primarily referred to "shifts" monitoring the equipment but now increasingly includes software work.

These libraries are a substantial achievement, providing a platform for the many analyses undertaken throughout the collaboration. As such it is software that is written to be used over and over again by what are essentially End User Programmers. It is subject to much more formal software engineering practices, such as tests and line by line code reviews. It is also documented for use in a way very different to the description of the analysis scripts in the tech. note. Despite its importance ROOT4STAR is not mentioned or cited in our focal paper.

Data and Simulation Production Software is involved two aspects of the analysis: the production of the initial dataset and the production of simulation data used to demonstrate the sensitivity of the analysis.

Data production refers to the shared processing of raw instrument readings into annotated datasets describing the first-level physical phenomenon in terms of events, for example particles and their energies and tracks. This can be highly computationally expensive, usually drawing on specialized clusters and on federated Grid computing resources. This work is undertaken through service work from collaboration members with substantial contribution from central IT. While the software involved in this work was not cited in the paper the Open Science Grid is acknowledged.

Our focal paper used a particular simulation technique called "embedding." Here a simulated target signal is embedded in the real background from the data and the analysis scripts run. This demonstrates and measures the ability of the analysis to identify the candidate signal in the real context of the instrument. The collaboration runs a central facility to provide embedding simulations for the publications of the collaboration. This is staffed in part by central IT staff but also with lab members performing "service work."

In the case of our focal paper the embedding simulation called for something new and specific, beyond the existing code of the central simulation production team. One of the two scientists writing the analysis scripts discussed above worked with the simulation team to build new code for the embedding simulation. The interview made it clear that this assistance was important to removing a bottleneck and getting their analysis to a publishable stage. This code was stored separate from the analysis scripts in the central simulation team's CVS because it was viewed as likely to be reused in the future. The embedding code drew on two packages produced by academics outside the collaboration: PYTHIA and GEANT. While the simulation code, as with ROOT4STAR and the data production code was not directly cited, these external packages were.

"Service work" offers an opportunity to contribute to the collaboration in important ways. It is provided by labs as a condition of membership, motivated by access to the data and inclusion on all authorship lists. It is also motivated because some additional grant funding is available for software work in the collaboration. One central administrator spoke of a concern with this situation saying that members sometimes "promise software but hope for science"; that the resources intended for software are cross-subsidizing work viewed as more directly linked to the collaboration's scientific mission.

CASE 2: STRUCTURAL BIOLOGY

Our second case is Structural Biology, a science concerned with the identification and characterization of molecules. There are many such molecules yet to be described, some of which prove very valuable in applied settings, especially for drug development. We interviewed five scientists and one information technologist working in this field, which is made up of many separate labs competing for primacy in analyzing the structure of their chosen molecule. This primary unit is reflected in the author lists of publications, typically between three and six authors. The lab head is listed last and the first authors are those primarily responsible for the specific analysis.

Focal Publication Our focal paper contributes to the "rational design" of drugs to counter botulism and was recently published in PLOSone. The study reveals the physical structure of peptides which form complexes with the Botulism toxin which provides crucial input to the process of developing drugs to block the disease agent. We interviewed the lab PI and the first author of the paper who confirmed this as "typical" of their work.

The work of describing the structure divides into two phases: a data collection phase primarily composed of wet lab work and a dry phase primarily composed of analysis supported by software. The wet lab phase involves isolating the molecules of interest, purifying them and crystalizing them. These crystals are then subjected to X-Ray Crystallography at a facility known as a Synchotron. This process was simplified in a description by one informant as bombarding the crystalized molecule and producing a set of x-ray shadows, followed by a software supported combinatorial search for known components of molecules whose structure and configuration might throw parts of this shadow. Our informant estimated the time spent on the project as two months, with 70% of time spent on wet work and 30% of time on the software-assisted analysis.

Identifying Software Involved

The first task requires obtaining the raw data from the Synchotron and converting it to a set of reflections and coordinate data. This involved two pieces of software (Scalepack and Denzo) available in a package called HKL-2000. The next step takes these inputs and produces an initial structure that could have caused the observed xray shadows. The author used a program called Phaser, included in a package called CCP4. This initial candidate structure is then interactively refined using a combination of visualization and fit statistics. The author used programs called PHENIX and COOT and produced a fit summary using a program called MolProbity. Finally the paper includes many illustrations of the structures as figures; these were produced with the PyMol package. Each step in this workflow is run manually. The final structure is, as a requirement of publication in Structural Biology, uploaded to the Protein Databank. The computing was all carried out on the lab cluster, with the exception of MolProbity which is available as a Web Service. The software was installed and maintained outside the specific lab, through an organization called SBGrid.

SBGrid SBGrid is a "small IT organization funded exclusively by our [140+] member laboratories" based at Harvard Medical School. It provides two services. The primary service is a distribution of software packages relevant to Structural Biology (the other service, not used for our focal paper, is access to the Open Science Grid). Members pay a fee (a few thousand dollars) typically funded from NIH grant money, but waived for a small number of labs which contribute a package to the distribution. The distribution was started by the SBGrid PI in 1999 to synchronize the software at multiple labs at which he was working. The PI is now a research-focused Associate Professor and the software distribution is managed by one full-time information technologist ("philosopher-cum-software-engineer") whom we interviewed. His full-time salary is paid from fees from the member labs.

The SBGrid software distribution works in a way very similar to linux software distributions, such as Debian or RedHat. The distribution bridges between software authors and users, providing pre-packaged, pre-compiled software applications arranged for maximum compatibility. The distribution also provides an updating service which pushes new versions out to member labs' computer systems. Further the coordinator monitors the projects producing each of the packages for relevant updates or bug-fixes. Finally SBGrid provides first level support for packages and helps members contact project developers for further support.

The author of our focal paper indicated that SBGrid saved a significant amount of time and gave him confidence that the packages were up to date and likely to work well together. Nonetheless SBGrid was not acknowledged or cited in the paper and the authors did not consider that necessary. SBGrid, in fact, does not provide a citation that could be used, nor request acknowledgement. They do maintain a list of member labs but do not maintain a list of papers that use their distribution.

CASE 3: MICRO-BIOLOGY THROUGH BIOINFORMATICS

Our third case study is in the field of microbial biology using bioinformatic techniques. The broad challenges in this field are to identify organisms, understand the relationships between organism's DNA and functions and to study the evolution of these organisms. These functions may find commercially important applications, such as in processing of plant-matter into bio-fuels. As with Structural Biology, this field is also organized into small labs with the lab head as the PI listed last and the primary author(s) listed first on publications.

Focal Publication We interviewed an author of a bioinformatics paper recently published in the journal Science. The paper focused on Nitrogen fixation functions of a micro ecosystem in which leaf-cutter ants cultivate fungal crops. Our informant was familiar with the full analysis and was able to report on packages used in that paper and in papers he was working on with a similar workflow.

The analysis begins with data collection to obtain relevant organisms from interesting systems. In this case this means field work in Argentina, Costa Rica and Panama. The organisms are cultured, generating sufficient DNA for sequencing at specialized centers, often supported by science funding agencies. The first software-dependent step is interpreting raw sequencing data as strings of DNA bases (A,C,T,G). The remaining steps all occur through comparisons between that sequence and public databases of previously sequenced and annotated DNA. Organisms are identified through a particular locus called 16S. Their full sequences are then compared to DNA databases which contain annotations regarding functions associated with different loci which, combined with knowledge of the researcher's particular system, enables reasoning about the likely functions of the organism. Finally evolutionary trees are constructed.

Identifying software involved

Identifying the software used for this publication was relatively simple. During the interview we were referred to the Supporting Online Materials appendix to the paper. While the paper itself is 3.5 pages long, the appendix is 35 pages long. The first 6 pages are Materials and Methods, 12 pages of "Supporting Text," 17 pages of detailed data and 2 pages of references. The Materials and Methods section takes care to identify each software package used, using different kinds of citations which we discuss below.

Interpreting the raw sequence data was done with the Sequencher package, 16S identification with the ARB and Greengenes package, full sequence alignment with the muscle package and the NCBI BLAST webservice and phyllogentic trees constructed using the MrBayes package. Beyond these packages, our informant also described a series of "power-user" scripts he wrote that linked together packages like these in similar papers.

Sequencher The sequencing in this case occurred via a chromatagraph, which exploits differing rates of fluorescence between the four components of DNA. The raw flourescene data must be interpreted in a software-dependant process. This software comes in two forms: either interactive, used for short sequences (such as 16S) or fully automated for whole genome sequencing.

The interactive software used in this lab is called Sequencher, provided by a specialized company (Gene Codes Corporation). We were not able to interview its creator, but know something of its background from media coverage. Its author, and now CEO of the company, was originally trained in music and psycholinguistics. He formed the company in 1988, releasing the Sequencher package in 1991. In 2001 the company was asked to help with the identification of the remains of those killed at the World Trade Centre through shotgun DNA sequencing (separating DNA from multiple sources) and the firm now markets "Forensic DNS analysis software" in addition to Sequencher.

The incentives for producing this software are straightforward: it is a profit centre for the firm. Sequencher is a fully commercial package; it must be paid for both academic and commercial use and license management includes key servers and dongles. We believe that the software is written and maintained in house at the firm's Ann Arbor Michigan office by employed software engineers.

The author on our focal paper indicated that Sequencher is trusted because it is "widely used" and is the standard software for interactive gene sequence construction. The software is cited in text with just the name of the firm; it does not appear in the References listing. In this way it is treated in a manner most similar to providers of hardware such as pipettes and chemicals.

Newbler An example of automated software for whole gene construction is Newbler, which is written by a "next generation" sequencing machine company (454 Life Sciences). The software runs directly on 454 Sequencing machines, but is also made freely available to researchers, including source code for advanced configuration. Our focal paper does not name the software directly but describes the equipment, naming the firm and citing a Nature paper. There are 45 authors for this paper, employees of 454 Life Sciences and academic scientists and aims to "describe a scalable, highly parallel sequencing system with raw throughput significantly greater than that of state-of-the-art capillary electrophoresis instruments." The incentives for producing and maintaining this software are clearly linked to the sale of the 454 machines; the software is essentially useless without data derived from these machines. The Nature publication is likely of great value to those listed with academic connections or aspirations.

As with Sequencher our informant indicated that this software is trusted because it is widely used and associated with a highly reputable company.

Greengenes and ARB Greengenes is both a specialized database for 16S sequences and a specialized web service for organism identification. Both the database and the software are produced, maintained and made available as a service by the Lawrence Berkeley National Laboratory. The service is freely available for all users, including commercial use. Our focal paper provides a citation in the reference list to the Applied Environmental Microbiology journal; the home page for the software has "Citation" as a top-level item and lists this publication first. The homepage has a short "Those citing Greengenes section" which lists only three papers; our informant indicated that it was much more widely used than this.

ARB is a GUI client focused on 16S identification tools. It is produced, maintained and made available by two German academic institutions. The paper provides a citation to a publication titled, "ARB: a software environment for sequence data" in Nucleic Acids Research and all authors are associated with the two academic institutions. The download page states that "Use is permitted for non-profit purposes" but does not contain clear conditions or purchase information for commercial use.

Our informant indicated again that both these packages are trusted because they are widely used and from reputable sources.

BLAST Webservice The final three steps compare the sequences with known sequences by comparing against public databases. This process involves an algorithm known as BLAST, in the words of our informant, "BLAST is the most important and most useful and most used piece of software in biology." This algorithm finds appropriate matches in the strings of DNA sequences. The algorithm itself is implemented in many packages, including ARB and Greengenes. The publication describing BLAST is now the most cited paper in all of biology.

The National Centre for Biotechnology Information, funded by NIH, provides a Web Service version of BLAST which accesses the US Genbank as well as downloads of a canonical implementation of the BLAST algorithm. The NCBI maintains and provides BLAST as a "top-level" service along with the PubMed library. We have not interviewed the maintainers of the BLAST web-service, but our informant indicated that he believed the BLAST software and webservice to be maintained by professional software engineers under the guidance of scientists employed by NCBI.

Muscle For long sequence alignment they used a package called MUSCLE. In the focal paper this is cited with an academic citation to a single authored article published in Nucleic Acids Research. We interviewed the author of the Muscle package and the paper. This package is relatively well-known; the author points to over 10,000 citations to the two publications describing the package. However, it is not as widely used as its "slick GUI" competitor ClustalW.

The author describes himself as an "unemployed gentleman scholar," a phase best clarified by a short biography. He earned a Physics PhD, beginning a postdoc but deciding that he was a better "software person" than he was a Physcist. In the early 1990s he began a voicemail technology company which he sold to Intel in 1999. The proceeds from this sale have supported him through until now. Following the sale he was "looking for interesting problems" and attended some biology lectures at nearby UC Berkeley where he came to the realization that, in his words, "biology == strcmp()" [the C string comparison function name]. He worked with a professor over a summer and produced the forerunner of Muscle, releasing the package itself a short time later. The package is released as public domain, which he believes to be more appropriate than open source licenses.

He is now well published in this area, on the editorial board of relevant conferences and is one of a handful of contributors he refers to as "algorithm people," distinguishing them from biologists. Recently he has produced an algorithm he describes as 500 times as fast and good as BLAST for alignment of very long sequences. This he now releases under a mixed license, closed source but free for academics and available for a fee to commercial users. The use of such a hybrid setup is new for him.

He characterized the environment for the production of these tools as highly individualized and competitive. He does not build on other tools in his domain, nor does he work to push his contributions into broader packages. He argues that this enables him to be unconstrained by architectures and existing code and because it promotes code portability (by not requiring libraries etc). He considered it obvious that algorithm writers would not collaborate. Pushed, he offered two reasons for this. The first was that there was little credit to be had in being seen to incrementally improve another's package, and any contributions—even substantial re-writes—would accrue citations to the original author's papers, even if he "published on" the changes. The second was that he did not wish to be at the mercy of members of other teams in decisions about what to include in packages; this would not only be slow and involve substantial politics but would be inappropriate given the competitive nature of this field.

Private "power-user" scripts Our Structural Biology informant described himself as a power-user meaning he preferred speed and accuracy (cli) over ease of use (GUI). It also means that he is comfortable working with independent tools rather than integrated software suites, which allows him to choose individual tool most suited to a particular task. To make these tools work together our informant writes scripts which contain configuration settings, convert to appropriate data formats and execute other tools. These personal scripts allow our informant to orchestrate these tools for multiple different analyses. Scripts such as these were not mentioned at all in the paper or the Methods and Materials section of the paper. They do, however, appear to form a type of comparative advantage for our informant, who related being able to perform—in a weekend—preliminary analyses which had been out of reach of his lab prior to his arrival.

Our informant held it as axiomatic that a paper, in the methods and materials section, should provide sufficient detail for its replication. However, he considered it only a limited responsibility to assist anyone attempting such a replication; they must "do their homework." He did not consider it likely or appropriate for readers, or even reviewers, to seek access to his personal scripts. He considered these scripts to be quite different from other types of software. If he had written a novel tool embodying a non-intuitive technique and used it in a paper, he would feel responsible for releasing that.

The author trusts these scripts because he wrote them himself, they are simple workflows, and he constantly compares their results to known scientific parameters; he argues that he is easily able to see if the results are nonsensical.

DISCUSSION

The three cases above allow us to abstract a set of production systems through which software used in science is created, maintained and shared. These systems are primarily differentiated by the rewards available and which act as incentives to motivate activity. Each system has associated with it a set of software engineering practices and produces software with differing logics of correctness.

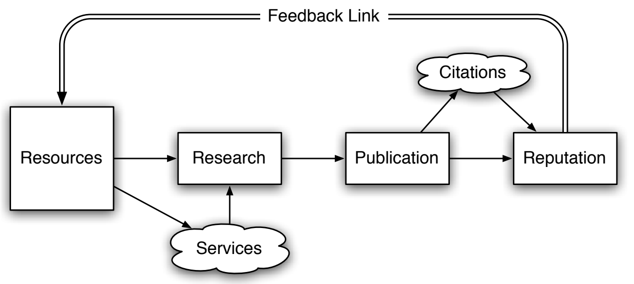

Figure 1. An archetypal simplification of the scientific reputation economy

The link between practices and the incentives that motivate them can be in a variety of configurations. Moreover these configurations have strong effects on the production of software in science. In some circumstances, we will argue, these can be out of alignment and result in sub-optimal situations for the production of software for science.

Software is a secondary player in the world of scientific work, which is dominated by a reputation economy based on substantive scientific publications. Figure 1 attempts to illustrate a much simplified archetypal scientific production system. Research realized in publications is the core activity of scientists. Research, of course, requires resources, realized through salaries and, especially, grant funding. Unlike commercial production the core activities do not directly result in the resources, since publications are given away for free. Obtaining resources, then, is indirect through reputation, and the path from publication to reputation is direct and quick. If the publications are useful to other researchers, then the publication receives citations, boosting the reputation of the authors. Publications do not need to be maintained as a separately motivated activity. Providers of essential services and equipment, from the perspective of the research economy, stay very much in the background, perhaps rising to being named in the acknowledgement section, or in the case of scientific equipment suppliers, their product names perhaps mentioned in the methods and materials appendix.

Software as Supporting Service

Some scientific software falls comfortably into the supporting service category. This software earns its keep essentially outside the reputation economy of science. In our cases we saw three different types of this kind of software. The first is entirely commercial production, including packages such as Mathematica or SAS and, from Case 3, Sequencher. The second type was SBGrid from Case 2, where useful software work is funded as a service by membership fees paid by contribution or from grant money. The third covers the software professionals employed by large collaborations seen in Physics, those working full-time on software, rather that "service work" (which is considered below).

Software produced as a support service is distinguished because they do not typically receive mention in academic reference lists, either because they do not publish academic papers or authors do not perceive them as requiring citation.

This was the case in our papers for Sequencher, SBGrid and the ROOT4STAR framework.

Software for academic credit

The second major software production system identified in this paper is unique to science: academic credit as incentive. This breaks down into two types: the first is software that is indirectly rewarded with academic credit since it facilitates science publications, the second is software released for credit on its own which breaks into two types. The first is building a parallel academic software practice. The second is for a field to recognize software publications as scientific publications in their own subfield.

Incidental software Some software is written purely to facilitate research. Examples from our cases include the analysis scripts from Case 1 and the power-user scripts from Case 3. Their production and motivation is relatively unproblematic: they are useful because they facilitate a specific piece of research and worked on and adapted when (and if) another specific piece of research calls for it. This type of software is typically not made available for others to use, at least not in any formal or on-going way. It might be archived and provided to scientists interested in that paper on request, but without the expectation that the software will require ongoing maintenance work by academics. This type of software is typically written by individuals and evolves in the context of the science project. Its logic of correctness is that its author is close and knows it well, and that it performs within expected scientific parameters. These scientific parameters in effect provide a type of informal software requirements and iteration and discussion of results a type of informal software testing. This is closest to Segal's "professional end-user developers" category [14].

A parallel software practice Figure 2 shows the situation facing many scientists who release software for others to use. The top of this figure shows the same main feedback loop as Figure 1: a flow from resources, to research, to publication. Here the publication is referred to as *topical* indicating that it addresses a research question in the scientists field. As above there is a direct link from publication to reputation and, over time, citations which increase reputation.

The bottom part of the figure shows what our interview subjects referred to as "publishing on the software" describing the software produced in the course of research in a separate software publication. Such publications also directly result in reputation, although it is perhaps a lesser or different form of reputation. At worst it could be viewed as evidence of distraction from the scientist's main line of work. Nonetheless whatever reputation is earned form the publication is direct.

Earning citations for software, however, is more circuitous. Unlike topical scientific publications the software publication is rarely useful by itself; it is generally the software that is useful to other scientists. The use of that software package by others in the future garners citations to the software publication. In this figure this link is marked with a dashed line to indicate that this linkage is more uncertain than regular pub-

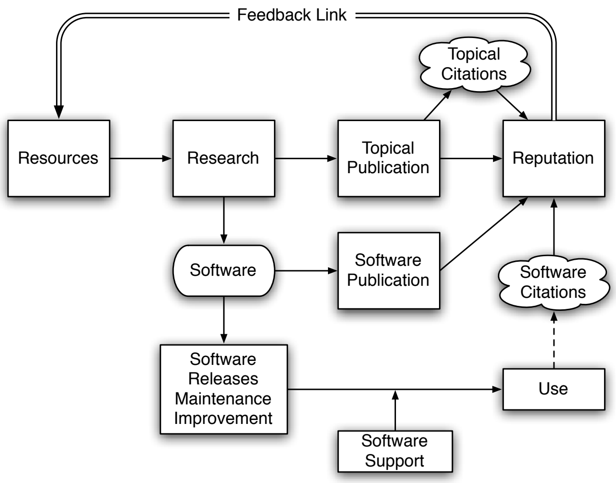

Figure 2. A depiction of the reputation incentives in a mixed science and software academic practice

lications. This is because the ethical requirement for citation is less well established; not all users of scientific software view it is required to cite the software, and not all software packages make a convenient citation available. We believe that it is possible that this results from authors viewing the software as a support service, as in Figure 2 and assuming that it is rewarded outside the reputation economy.

Making software useful for others is more complicated than with topical publications or even dataset publications. Software, even after being "published on," must be released for other's use. Such sharing is costly. Effort must be made to make the package generally useful and secure for release; in Case 1 our informants told us that an open source requirement for internal collaboration code would require costly security audits. Moreover, software requires maintenance effort if it is to remain useful in a constantly changing information technology environment. This would include, for example, ensuring it continues to work with newer versions of libraries that it uses, that it is capable of working with newer data formats and so on. Such maintenance is not going to be sufficient for a new software publication, but is necessary if the software is going to continue to provide reputation payoffs. Finally, the authors may find that they have to provide support to users. Software publications do not help much, since they are quite different from user manuals. They typically do not comprehensively codify the knowledge required to garner further use and further citations in the future. A software release thus carries significant costs, but is likely necessary to garner citations and draw reputation from an active userbase.

In a very small number of cases, illustrated by BLAST above, the software might become so important that these responsibilities of maintenance and support are taken off the author's hands and become a responsibility of a software professional, funded directly through science funding bodies.

In this way a dual science and software practice is a complex and questionable proposition for working scientists. Even if the software is created in a way that is required for their topical publications (and thus its creation is no extra effort) if the scientist seeks to build independent scientific reputation for their software the path is complex and the additional effort substantial. This additional effort takes time away from that which could be invested in topical work where the payoff is more clear. Finally there is the possibility that a scientific software practice enhances topical scientific competitors' abilities in a way that reduces the comparative advantage of the software's author, further decreasing their ability to earn topical reputation. In a reputation economy where topical publications are king, this illustration makes the difficulties of scientific software clear. With this incentive structure, it seems highly likely that sharing and maintenance of software is likely to be under-resourced.

A software subfield The second production system associated with direct academic credit is the development of a software subfield. This appears to be the case in computational biology, where publications about software are viewed as primary contributions. For example Carnegie Mellon University now has a Center for Computational Biology housed in the School of Computer Science, illustrating the centrality of software to this field. This takes the archetypal flow of science shown in Figure 1 and replicates it with software as the research subject. The formation of departments with this focus means that the institutional judges of reputation are appropriately aligned with the need for software contribution. Nonetheless the separation between a software publication and working software described for the parallel practice persists. Software and algorithmic publications advance the theory but someone has to maintain and support the working software.

Hybrids

We also identified hybrids between the two major production systems identified above. The first is a hybrid between direct academic credit and commerical software production. The second is a hybrid between cooperative support service and indirect academic credit, seen in the large Physics collaborations of Case 1.

Dual-licensing Commerical/Academic hybrids place a foot in the commercial economy and a foot in the academic reputation economy. The Muscle author's dual licensing direction in Case 3 illustrates this approach. Software licenses can be written so that resources are gained directly from commercial use, while academic publications and use garners citations and builds an academic profile. The commercial resources cross-subsidize the academic requirements of maintenance and support. Some of the commercial packages considered above appear to have begun in this mode. This is a well-aligned situation, generating financial resources for costly releases, maintenance and support that also supports academic use. However clearly this is only possible in areas where there is commercial demand for the software and this is not the case for much scientific software.

Collaboration Service Work Figure 3 illustrates the hybrid innovation represented by the physics collaborations

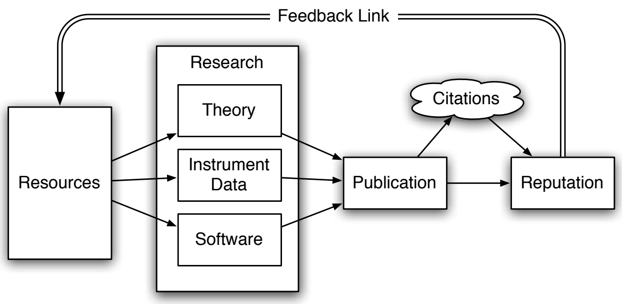

Figure 3. Depiction of changes to scientific reputation economy in large Physics collaboration

discussed in Case 1. Here the basic contours of the archetypal system remain unchanged, but the category of Research is widened to include many more of the supporting activities, including theory, the design, building, operation and maintenance of the data generating machines and the production and maintenance of software. This includes software work provided to the collaboration as part of the "service work economy" described above. All of the collaboration members are listed as authors, regardless of whether they worked on that specific paper. At first brush then, this appears to produce an appropriate alignment of incentives, with some collective earning reputation and converting it to resources available to all.

However, our interviews show that this arrangement pushes the question of rewarding software work back inside the collaboration. Externally the collaborations maintain that all contributions are equal and ought to be respected as such by external institutions. This is not unproblematic as illustrated by an academic associated with the DZERO collaboration: His university's tenure requirements include printing out copies of all papers on which one is an author. He diligently compiled all the DZERO publications and submitted them to the department secretary for printing and adding to the tenure binder: over 2,000 pages later the printer had broken. It is unlikely that the regular tenure case process can operate with these publications as evidence of contribution.

Our interviews confirm this: internally reputation is garnered differentially and assigned informally, through recommendations and visibility opportunities, such as opportunities to present preliminary posters. In this process, our informants indicated that software work by scientists, as service or towards papers, took a lower status position to instrument building but especially to theoretic and empirical scientific results.

Collaboration implications

The bifurcation between reputation for a software publication and citations derived from software use, shown in Figure 2, has potentially negative implications for direct collaboration on software between scientists. The authors of the initial software publication are frozen in time at its publication. Contributions to maintenance and support by others, while crucial for the software's usefulness and citations, are not rewarded by citations to the original paper. This means that they are hard to turn into sources of academic credit for the later collaborators; it is awkward to show on a resume or argue at a tenure case that you should derive some portion of the credit from citations to a paper on which you are not an author.

Some projects approach this by releasing additional publications, adding authors who have made significant contributions. However in order to have this paper published in a adequately high status journal, those contributions would have to be substantial. Regardless, users of the software may continue to cite the original paper. Making matters worse, if the contribution was significant enough for its own publication it is quite likely that the contributors would garner more reputation from creating their own package and publishing independently, as argued by the author of Muscle.

Another approach is to request that users cite "the software project" directly, rather than a publication, listing the authors and contributors in a separate, updatable location, similar to open source projects where files such as THANKS maintain contributor lists. This is awkward since such citations are sometimes difficult to add to bibliographies and instead relegated to footnotes. Even if they are added to bibliographies since they are not publications they may not be considered subjects for citation counts. Further, if a contributor arrives later in the project it is awkward to make the case for the size of the fraction of credit they ought to receive.

Given these issues it seems likely that significant software contributions to existing scientific software projects are not likely to be rewarded through the traditional reputation economy of science. Together these factors provide a reason to expect the over-production of independent scientific software packages, and the under-production of collaborative projects in which later academics build on the work of earlier ones.

CONCLUSION

Software is increasingly important to science. Modern scientific results depend on a network of software created inside and outside the direct context of the research. As opposed to hardware enables of science, software provides unparalleled opportunities to share and collaborate. Just as science results stand on the shoulders of those who developed new methods, validated instruments and theory, the software work in these collaborations draws together the combined work of many in the production of new science results.

This software is created in a variety of production systems, as described in our Discussion. These systems have differing alignments between the software, and software quality, expected from them and the rewards available within the system. While some of these seem relatively unproblematic, such as commercial production in fields with immediately valuable applications, others appear problematic. In particular we highlighted the potentially pernicious implications of the academic credit production system for collaboration and maintenance.

REFERENCES

1. J. Carver, R. Kendall, S. Squires, and D. Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Minneapolis, May 23-25 2007.
2. J. N. Cummings and S. Kiesler. Collaborative Research Across Disciplinary and Organizational Boundaries. *Social Studies of Science*, 35(5):703–722, 2005.
3. T. A. Finholt and G. M. Olson. From laboratories to collaboratories: A new organizational form for scientific collaboration. *Psychological Science*, 8(1):28–36, 1997.
4. L. Hatton. How accurate is scientific software? *IEEE Transactions on Software Engineering*, 20(10):785–797, 1994.
5. C. Hine. Databases as Scientific Instruments and Their Role in the Ordering of Scientific Work. *Social Studies of Science*, 36(2):269–298, 2006.
6. D. Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):119–120, 2007.
7. K. Knorr-Cetina. The ethnographic study of science: towards a constructivist interpretation of science. In K. Knoor-Cetina and M. Mulkay, editors, *Science Observed*, Beverly Hills, 1983. Sage.
8. K. A. Lawrence. Walking the tightrope: The balancing acts of a large e- research project. *Computer Supported Cooperative Work*, 15(4):385–411, 2006.
9. G. M. Olson, D. E. Atkins, R. Clauer, T. Finholt, F. Jahanian, T. I. Killeen, A. Prakash, and T. Weymouth. The upper atmospheric research collaboratory (uarc). *ACM Interactions*, 5(4):48–55, 1998.
10. D. Ribes and T. A. Finholt. Planning infrastructure for the long-term: Learning from cases in the natural sciences. In *Proceedings of the Third International Conference on e-Social Science*, Ann Arbor, MI, June 2007.
11. R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July-Aug. 2008.
12. J. Segal. Some problems of professional end user developers. In *Proc. IEEE Symp. Visual Languages and Human-Centric Computing (Vlhcc 07)*, pages 111–118, 2007.
13. J. Segal. Models of scientific software development. In *Proc. 2008 Workshop Software Eng. in Computational Science and Eng. (SecSe 08)*, 2008.
14. J. Segal. Software development cultures and cooperation problems: A field study of the early stages of development of software for a scientific community. *Computer Supported Cooperative Work (CSCW)*, 18(5):581–606, 12 2009.