Incentives and Integration In Scientific Software Production

James Howison
University of Texas at Austin
Austin, TX
jhowison@ischool.utexas.edu

James D Herbsleb
Carnegie Mellon University
Pittsburgh, PA
jdh@cs.cmu.edu

ABSTRACT
Science policy makers are looking for approaches to increase the extent of collaboration in the production of scientific software, looking to open collaborations in open source software for inspiration. We examine the software ecosystem surrounding BLAST, a key bioinformatics tool, identifying outside improvements and interviewing their authors. We find that academic credit is a powerful motivator for the production and revealing of improvements. Yet surprisingly, we also find that improvements motivated by academic credit are less likely to be integrated than those with other motivations, including financial gain. We argue that this is because integration makes it harder to see who has contributed what and thereby undermines the ability of reputation to function as a reward for collaboration. We consider how open source avoids these issues and conclude with policy approaches to promoting wider collaboration by addressing incentives for integration.

Author Keywords
Collaboration; software development; incentive systems; science policy

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

INTRODUCTION

Software is ubiquitous in science. In current practice, scientific software originates in a diverse set of production systems that differ markedly in their incentive structures for creating, using, sharing, and enhancing software [7,18,24]. A sea change is in progress, however, as science shifts toward cyberinfrastructure, offering to reduce duplicated effort and enhance large scale collaboration [3,9,33,43] both in science and in the production of scientific software. Science policy makers are seeking understanding of how best to shape the practice of science to enable this vision. An open collaboration approach has been successful in the production of open knowledge in Wikipedia and open source software [4,27,30]. Given the communitarian principles of science it is not surprising that open collaboration is a promising candidate [19,40,41].

This paper examines openness as a road to more effective collaboration and co-creation of scientific software. We report on results of an empirical study of the socio-technical structure of innovation around a central piece of infrastructural software, the Basic Local Alignment Search Tool (BLAST). BLAST is arguably one of the most important pieces of scientific software ever written, to a large extent enabling the bioinformatics revolution. By 2003 the original paper describing the BLAST algorithm had become the third most cited paper of all time [38].

Understanding the shifting and sometimes conflicting incentives for sharing and collaboration in scientific software will deepen our understanding of coordinating collaborative work more broadly. This is because "the republic of science" [29] contrasts with environments where open collaboration has previously been examined. The contrast is especially clear in terms of overall incentive structure. Free revealing through publication, thereby building long-term academic reputation in science, contrasts both with the financial incentives common in for-profit environments, as well as the combination of use-value, learning and localized reputation motivating open-source participation [e.g., 37]. By examining the effect of different kinds of incentives, this paper advances work on incentives in large-scale collaboration, a key theme of research in CSCW [e.g., 21].

BACKGROUND

Studies of software work in science/infrastructure

Research in CSCW has recognized the importance of "the human infrastructure of cyberinfrastructure" [25], including the environment in which those producing cyber-infrastructure work. Researchers have, for example, identified a tension between the academic research goals of computer scientists and the implementation oriented needs of domain scientists [14,23] and the need to "synergize" efforts in production of cyberinfrastructure tools, especially in shifting, dynamic funding environments [7]. In the related area of data, research has shown how differential valuations of data in different scientific fields result in varying incentives, which are key to understanding data-

CSCW '13, February 23–27, 2013, San Antonio, Texas, USA.

sharing practices in different scientific fields [43]. Recognizing academic publication as a primary motivation is key to understanding the challenges of collaborative production of scientific software [18]. Perhaps glossing over these distinctive academic incentives, many have pointed to the success of open source software development and argued that it provides a model for sustainability of software development in science [1,15,20,41].

Incentives in Peer Production

Incentives for participating in open source software projects have long been a subject of study. Expending effort for no pecuniary return was initially seen as a puzzle, explained in early literature in fairly idealistic terms, as supporting a form of freedom of expression [39], in terms of use value or "scratching one's itch" [36], or as symptomatic of a "gift economy" [36]. As academic attention turned to the question of incentives, focus began to center on the desire for reputation, for example, as a labor market signal [26] that might lead to a better salary. Surveys identified a complicated mix of incentives, including the desire to build a skill, share technical knowledge, and participate in a new form of cooperation [16,22]. Studies looking at the larger incentive picture have shown a complicated mix of intrinsic, extrinsic, and use value motivations, with various effects on developers' output [37]. Increasingly, open source developers often participate as paid employees of commercial firms, who invest in open source for commercial reasons [45]. Recent work has argued that there can be separate and different incentives for different elements of open source work such as production of code, revealing code and integrating code back into the shared source tree [5]. In a similar vein, other recent work has shown varying incentives for joining, contributing to, and becoming committed to online communities more generally [21].

As the novelty of open source software development has worn off, and as the commercial and open source worlds continue to intermingle, reputation seems to have reemerged as a primary motivation, as new environments make it increasingly easy to observe and evaluate others' work and popularity. For example, the number of "followers" a developer has is viewed as a signal of status in GitHub [13]. Indeed, online reputation accounting systems have emerged (e.g., Masterbranch.com and Coderwall.com) making reputation even more visible and explicit.

Recent work in scientific software has also shown a complicated mix of incentives in which reputation features prominently [18]. Yet in the scientific world there is the added twist that scientists, who write most of their software themselves, tend to be motivated by scientific reputation, not reputation in the software world. A physicist writing software gets little or no reputational credit unless the software enables physics work that enhances one's reputation as a physicist. This shift—working in one domain (software) while experiencing reputation incentives in another (science)—can produce mismatches [18], and may lead to patterns of activity that do not accord with those observed in non-scientific contexts. Moreover, these mismatches may lead to situations that do not serve the software needs of scientific communities well and provide difficulties for the vision of shared and communally-developed software infrastructure driven by open peer production.

A study of the BLAST innovation system

To explore these issues we studied the development and improvement of BLAST [2]. BLAST is an important piece of scientific infrastructure, realized in software code, which performs the lookup of primary biological sequences, including DNA and higher-level sequences including amino acids that make up proteins. These sequences are matched against annotated databases, especially GenBank, returning similar sequences based on appropriate statistics. These lookups facilitate scientific results including the identification of organisms, assessing gene function and structure and, in combination with other tools, analyzing their likely evolutionary origins.

Our specific research questions are the following:

1. Who created, maintains and improves BLAST?
2. What code was integrated with BLAST?
3. What motivates the creation, release and integration of improvements to BLAST?

Data and Analysis method

The results discussed below are based on an analysis of semi-structured interviews and inspection of published literature. We identified those involved with BLAST through a combination of searching in the published scientific literature, internet searches for versions of BLAST, and through asking each interviewee to identify participants and provide feedback on our searching strategy. Conscious of the possibility of selection bias due to examining the published literature we specifically asked each interviewee if they were aware of unpublished BLAST versions (discussed below).

We conducted interviews with informants who were well-placed to provide insight into the origins of improvements to BLAST and to explain their motivations in undertaking their work. The interviews were semi-structured using a protocol with three sections: the background of the interviewee and the relevant project, the source and nature of the BLAST innovation and motivations for the interviewee's software work, including how they make their case for impact. We sought permission to record the interviews for research purposes and all but one participant agreed. One interview was conducted face to face; the remaining interviews were conducted over the phone. Overall we conducted seven interviews with eight informants, representing seven cases of BLAST improvements. We identified four other improvements of

BLAST through literature and web-searches but our requests for interviews went unanswered. In these cases we have examined published papers and respective websites (where these exist).

For all interviews we made contemporaneous notes and developed summary memos immediately afterwards. We then developed near transcripts of the interviews and expanded our memos. Simultaneously we were returning to the research literature and we met periodically to discuss the interviews in light of this literature. In this way we developed systematic analysis artifacts focusing on organizing the cases according to their incentives for different activities. These are summarized following each case and together in the discussion. As our results developed we actively sought alternative explanations, challenging our emerging understanding. These alternative explanations, together with the rich case narratives, provide transparency into our analysis, allowing readers to see and judge the logic by which we arrive at our conclusions.

In the remainder of the paper we first report qualitative narratives of cases of BLAST improvement, then summarize motivations across the cases at each stage (development, revealing and integration). In the discussion that follows we consider alternative explanations of these findings (generality of solutions, integration costs, intentions and incentives) in their best possible light. We argue that one explanation, based on incentives, is most consistent. We then consider the theoretical and practical implications of that explanation.

RESULTS

NCBI BLAST

BLAST was conceived as an algorithm and piece of software by a small group of academics at the National Center for Biotechnology Information (NCBI) together with a tenure-track biologist at Penn State and a tenure-track computer scientist at the University of Arizona. The NCBI was funded by the National Library of Medicine, an Institute of the National Institutes of Health, and managed the GenBank DNA database. The software continues to be developed and maintained primarily by employees of NCBI most of whom have PhD backgrounds in Mathematics, Computer Science and Biology. They have shifted career tracks to software work in the service of science, but participation in science remains an important element of their professional identity (as opposed to an identity as software developers who incidentally work in a science domain).

Our interviews revealed that the creation of BLAST was motivated by a combination of use value (for the working biologists in the group) and academic credit (all authors sought to publish to build academic reputation). Ongoing maintenance is motivated as a service project for science. The developers of BLAST expressed interest in following the improvements of BLAST created outside of NCBI, and have integrated some outside contributions (see below) but have not sought explicitly to encourage the development of BLAST as a community project.

BLAST+

BLAST+ is a complete rewrite of BLAST by the core BLAST team at NCBI. It was released in 2009 and described in a contemporaneous academic publication [10]. The paper argues that the inclusion of new features over time had undermined the BLAST source code: "the continual addition of unforeseen modifications made the BLAST code fragile and difficult to maintain." [10] The primary goal of the rewrite was that, "the code structure should be modular enough to allow easy modification." In addition the re-write process enabled the NCBI authors to add features, including some that had been demonstrated by external improvements and provided by forked code versions hosted outside NCBI for some time (FSA-BLAST and MegaBLAST). The BLAST+ codebase provides substantial improvements to most, but not all, BLAST operations and is now the primary non-historical version of BLAST available from NCBI.

The authors of BLAST+ were motivated by use-value, although it was a different use-value than the use-value indicated by working biologists. The use-value here was to make the NCBI's overall task of service provision to the bio-informatics community more efficient and effective. The production of an academic publication to describe the code was primarily justified as the appropriate way to announce and document the new code, but since the NCBI operates in a quasi-academic environment there was a secondary motivation of academic credit associated with the article publication.

WU-BLAST

BLAST, as originally released, lacked the ability to conduct searches with gaps in the sequence. WU-BLAST was the first package to provide a "gapped" BLAST. Its author had originally been employed at NCBI to work on NCBI BLAST after having been recruited out of a programmer-analyst position at UC Berkeley. Seeking a more academic track, he left NCBI for a tenure-track biology position at Washington University in St. Louis where he intended to continue to improve BLAST as well as use it to analyze data produced by the WU Genome Sequencing Center. WU-BLAST was maintained as a separate project with performance that earned it a loyal user base. Some of the features WU-BLAST introduced, including gapped searches, found their way into NCBI BLAST, although source code was never integrated. The author recently renamed the project AB-BLAST, forming a company and selling commercial licenses.

Digital/Compaq BLAST

As was common with hardware companies at the time, DEC maintained a vertical technical marketing group whose focus was to ensure that software relevant to their

market segment ran smoothly on their specific hardware. In 1999, Compaq acquired DEC but the Alpha business continued to run as it had under DEC. Our informant was a member of a technical marketing group. In the course of his work, he increasingly encountered potential and current clients engaging in sequence analysis. To serve this group the technical marketing group collaborated with an engineering group within the old DEC structure that had created an enhanced version of BLAST based on the NCBI code. The technical marketing group used that version to generate benchmarks showing the improved performance of the Alpha platform in running BLAST. These benchmarks, and the software improvements, were used to "sell hardware." Our informant argued that, from the perspective of DEC Compaq the incentives for development and maintenance were financial: they gained access to a market segment and sought to increase their income.

It was common practice at the time for DEC to provide enhancements to the originators of the software. Our informant indicated that this was important for two reasons: first it ensured that DEC was no longer responsible for maintaining a forked codebase as the software's original authors improved their code. Second, customers were "more confident" obtaining and using code provided by original authors. In the case of BLAST that meant customers preferred to use NCBI BLAST. In keeping with this practice DEC/Compaq reached out to NCBI and worked with them to integrate their enhancements into the mainline NCBI tree; NCBI acknowledges this contribution in the paper that describes BLAST+ [12].

Mac OS X port

In the late 1990s and early 2000s Apple transitioned to a Unix based operating system, known as Mac OS X. NCBI BLAST was ported to Mac OS X primarily by our informant. Our informant held a PhD in biology and on graduation had asked "what does a person do that enjoys biology and computers at the same time?" The majority of sequencing work at his workplace was done on DEC Alpha workstations that remained in the office and doubled as developer machines. Our informant, however, preferred to work on his Apple laptop. Yet the majority of the emerging bioinformatics tools at the time did not compile for Mac OS X out of the box, leading our informant to use time during his 1 hour daily train commute to port these applications. He thus became active in a community of like-minded enthusiasts who ported relevant applications.

Apple provided support to the porting community, including trips to Apple HQ in Cupertino and Apple computers. Documents show that Apple was planning to release their Xserve rack-mounted server and increasingly saw life sciences as an important market, both in academia and in industry.[1]

Our informant maintained the Mac OS X port for a year or two, providing binaries via FTP. Apple also distributed these binaries on DVD images at life sciences computing conferences to ensure that their customers had access to the code that worked on their hardware. Our informant expected that the projects he ported would integrate his changes, because he wanted to have impact and to reduce the support workload of making them available and updating them. He found himself "doctoring" each new release of BLAST from NCBI, sending his changes to an NCBI contact that he was introduced to by Apple, who also wanted the port integrated. Eventually he stopped being asked for the code and found his port had been incorporated in mainline NCBI BLAST, as part of the interaction between Apple and NCBI described below.

A/G BLAST

A/G BLAST was developed by Apple Computer in conjunction with Genentech, a life sciences company, as a lead user and collaborator. A/G BLAST was optimized by Apple to run on the G4 PowerPC chip, which included a proprietary "Velocity Engine" called "Altivec". As with DEC/Compaq, many of the improvements were not specific to Apple chips, but Apple emphasized the role of their proprietary chips in marketing materials.

Development of this BLAST improvement was thus motivated by complementary goods sales, i.e., making the software available in order to facilitate hardware sales. They hoped to ensure that the G4 chip with its Velocity Engine was used to its potential for this application. In the words of a contemporaneous press release introducing A/G BLAST Apple extolls the virtues of their hardware for science in a section entitled, "Velocity Engine and Mac OS X: The Ultimate Science Platform."[2] A/G BLAST was touted as superior, able to deliver "five times the performance of the nearest competing desktop system running standard NCBI BLAST." [2]

The source code for A/G BLAST was freely revealed and made available through Apple's websites as well as distributed at life science conferences. In a manner similar to the Digital/Compaq Alpha port, because the code only ran on Apple's hardware there was no effort to restrict the availability of this code. Moreover at the time Apple had a strategic push towards open source software and the open source community. The relevant group at Apple did publish academic papers from time to time, but did not publish a paper describing A/G Blast, nor otherwise attempt to earn academic credit for the improvements in A/G Blast. Rather Apple made the code available on an Apple-branded website and publicizing through press releases and presentations at industry venues, such as sites and conferences organized by O'Reilly Media.[3]

[1] Stewart (2001) "Bioinformatics meets OS X" O'Reilly Media. http://oreilly.com/pub/a/mac/2001/12/14/macbio.html

[2] http://www.apple.com/pr/library/2002/feb/07blast.html

[3] Originally hosted at http://developer.apple.com/hardware/ve/acgresearch.html and as of May

Apple sought to have its improvements moved back to the NCBI BLAST, which was, for Apple's customers, easier to justify using in academic papers. Moreover the A/G optimizations could be better kept in sync with the ongoing development of NCBI BLAST, reducing effort for Apple's programmers and ensuring that new features from NCBI were available to purchasers of the G4 Apple Hardware. The NCBI BLAST team initially became aware of A/G BLAST through publicity associated with the O'Reilly Bioinformatics conference and was eventually invited to Cupertino where engineers described the Velocity Engine and other optimizations. NCBI ported these to their main source tree soon afterwards. As with Compaq's contributions NCBI acknowledges Apple's contributions in their BLAST+ publication [12].

GPU-BLAST

GPU-BLAST is an effort to exploit the parallelism available in modern GPU platforms, specifically the NVIDIA GPU [44]. It was developed by two academics whose focus is primarily on computational optimization, and neither considers themselves biologists or bio-informaticians. Whereas previous attempts to implement BLAST on GPU hardware have provided significant speed-ups under some conditions they have not provided identical results; GPU-BLAST ensures an exact results match with mainline BLAST.

GPU-BLAST, unlike other BLAST improvements considered in this paper, was built on the re-modularized BLAST+ codebase. The GPU-BLAST authors indicated that the BLAST+ code was very well written and was relatively easy to modify. They indicated that they only needed the code itself and did not rely on documentation nor seek help from NCBI. Nonetheless they did not see that as a reason to base their work on the BLAST+ codebase, arguing that the original BLAST code was also well-written and easy for their modifications. They chose to base their work on BLAST+ to be up to date, rather than seeking to exploit the more modular and easier to integrate structure of the codebase. The authors moved quickly to publish a description of GPU-BLAST in the academic literature. As part of that publication they made their source code available, hosting GPU-BLAST on their own website. That website is very specific about their expectation that users would provide academic credit through citations to their publication: "Please cite the authors in any work or product based on this material."

The software is also promoted by the NVIDIA's GPU platform developer's program as part of a bioinformatics distribution. NVIDIA found the GPU-BLAST software as they prepared a bid to sell hardware to a large hospital and added it to this collection. Following this NVIDIA donated "a few cards" to the GPU-BLAST authors, much as DEC and Apple had before them. Other than this post-hoc donation GPU-BLAST authors receive no royalties or financial benefits from their code.

2011 still available at http://developer.apple.com/opensource/tools/blast.html, although optimizations were eventually integrated into NCBI BLAST. See http://www.apple.com/pr/library/2002/feb/07blast.html and http://www.xml.com/pub/r/1327

Other Improvements

We identified four additional improvements for which we were unable to obtain interviews: commercial internal improvements, and three improvements described in academic publications (CUDA-BLAST, FSA-BLAST and CS-BLAST). Multiple informants indicated that they believed that commercial users of BLAST had internal improvements which they did not release publicly, while our informants did not have specific inside knowledge they indicated their belief that these were motivated by financial incentives in that they provided competitive advantage to the firms.

Improvements in the second group were found through literature searching but we were not able to interview the authors. In each case the code was released simultaneously with publication through the author's own websites. Although not as reliable as reasoning based on interview data, we infer a desire to obtain academic credit from the authors undertaking publication of these articles. In the two older cases the code is no longer available at the links provided. However, as with the "gapped" improvement in WU-BLAST the core NCBI team has incorporated the ideas, but not the source code, of those publications.

Summary of motivations

Our informants discussed their motivations for the three activities of development, revealing and integration.

Development was motivated by a range of motivations. One driver was use value: the improvement was necessary to support work that the informant wished to undertake. Another driver was seeking academic credit: by developing an improvement that would be useful to the academic community informants hoped to receive reputational rewards that would advance their careers, including seeking tenure and promotion. Academic credit could be earned through improvements to the usefulness of BLAST, but also more general contributions, such as novel optimization techniques. In one case development was motivated because it was simply a fun and enjoyable activity ("there was no credit that I was competing for"). Finally, one group of improvements was motivated by a desire to sell complementary goods and thus increase revenue: if BLAST performed better on particular hardware then customers would be more likely to buy that hardware.

Revealing is the action of sharing the code with others. While one informant argued that they were ethically bound to reveal code, the other informants argued that revealing was crucial to access the benefits that had motivated development. This was true for those motivated by academic credit: getting the software to users, even if just in

binary form, is crucial to receiving academic credit and, given the norms of academia, source code revealing is usual (despite the possibilities it offers to competitors). For those seeking revenue through complementary goods, revealing was also necessary. The software already ran and was available on competing computing platforms; the optimizations were given away to enhance the perceived value of buying particular hardware; informants indicated that the code was released "via PR [public relations]."

Integration involves merging an improvement with the mainline BLAST. While any improvement could be made available, only integrated improvements would be distributed by NCBI. Integration also implied that NCBI would undertake to maintain the improvement going forward, accepting it into the scope of the service that they provide their community. For one group of informants these were both positive motivations to have their code integrated. By integrating their customers could have the knowledge that they were working with "the *real* BLAST" making them more comfortable with the correctness and acceptability of the results. While it was good for hardware sales to have an optimized code version, it was even better to have mainline BLAST optimized. Once the code was integrated, however, any on-going work, such as synchronizing with new versions, was a cost; if this could be borne by the BLAST developers that would increase net revenue.

For another group, however, while reduced costs of maintenance from integration would be welcome (NCBI BLAST was the "industry standard"), that benefit did not outweigh the perceived cost of integration. The question of who would receive credit for the combined artifact was key, either from direct ongoing use of the software or through academic publishing and citation. Since users would be citing a paper in their academic work, some informants expected that users would continue to cite the "original" BLAST paper, rather than the new publication of the improver. While the possibility of a joint publication was enticing, in one case it was suggested in our interview that there was a concern that the original BLAST authors would not benefit from creating a new publication that would undermine the citation of their existing papers.

RESULTS SUMMARY

As our cases proceeded we created systematic analysis artifacts that summarized our answers to our second and third questions: Was the code integrated? And, what motivated development, release and integration (if it occurred)? These artifacts bring our results together in a systematic manner and we present them in this section.

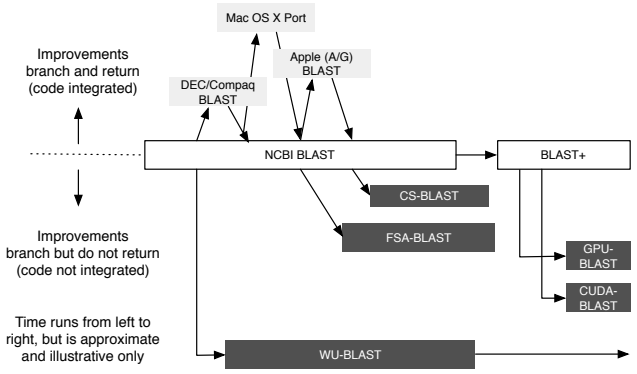

Figure 1: Forking and integration in the BLAST software ecosystem

Figure 1 summarizes the branching structure of the BLAST improvements we studied. At the core is NCBI-BLAST, transitioning to BLAST+. We found all improvements to be based on this line of code. Thus we found all our improvers sought to base their improvements on the core NCBI distribution, and to do so on whichever version was most up to date, transitioning from BLAST to BLAST+ when the NCBI did.

The path of these improvements can be divided into two basic structures: those that branch and return, being integrated with the NCBI code (shown above NCBI BLAST in light gray) and those that branch but whose code is not integrated (shown below NCBI BLAST in dark gray). Those that were not integrated were maintained separately for some time by their authors, typically having them available on personal websites. Our informants indicated that they intended to make their versions available and update them to be in sync with new versions from NCBI, but they were not sure how long they would do so; we found that forks tended to be maintained for relatively short periods of time and their authors, rather than announcing an end to the project appear to simply stop updating their websites.

What can explain the differences in whether the improvements had their source code integrated? Through our analysis we developed four competing explanations: generalizability, integration costs, intentions and motivational conflicts.

Generalizability. *The providers of a popular distribution may not want to increase the size and complexity of the distribution unless some large fraction of users will benefit.*

Improvements may be too specific to be generally useful., so including every improvement would complicate the codebase too much for too little gain. This provides a content-based explanation for our observed pattern: improvements that were generally useful would be integrated, while more specific improvements would not. Certainly the commercially motivated computing platform based improvements, which were all integrated, fit with this explanation: all uses, if not all users, of BLAST could benefit from a port to Mac OS X or enhancements by,

DEC/Compaq or Apple/Genentech. However there are quite general improvements in the cases where the code was not integrated, including gapped BLAST from WU-BLAST, the improvements in CS and FSA BLAST and the code from GPU-BLAST. Thus a content-based generality does not provide a consistent explanation for the observed patterns, nor was it emphasized by our informants.

Integration costs. *It takes technical effort—sometimes a great deal of technical effort—to integrate new functionality into a large, complex piece of software, and to maintain it over time.*

A plausible explanation for why some outside improvements came to be integrated while others did not is that some simply required more effort to integrate and were thus either not considered worthwhile or the improvement was incorporated by writing code from scratch. Code written by others can be impenetrable and it can be difficult to isolate the useful improvements. One key cause of high integration costs is related to the modularity of the codebase. The more modular a codebase is the lower integration costs should be [4]. Our informants indicated that the original BLAST codebase was fairly monolithic and that the BLAST+ codebase was clearly more modular (as intended by its authors). This explanation would thus predict that code based on BLAST would be less likely to be integrated than code based on BLAST+. Admittedly we have only one case of code based on BLAST+, but we have more cases based on BLAST. In neither condition, however, does this explanation appear to explain the observed patterns, in fact the opposite holds. All the cases of integrated code were originally based on BLAST, while the case based on BLAST+ was not integrated. If the integration costs explanation held this pattern should have been reversed. Given the small number of cases here, in our opinion it is more telling that our informants did not emphasize this explanation, focusing far more on maintenance costs than on integration costs. Either way integration costs does not seem a sufficient explanation.

Intentions. *Improvements were integrated unless their creator did not intend to have them integrated.*

The simplest explanation would be that the authors, as copyright holders, did not want their code integrated. However, in most cases the outside contributor clearly expressed a desire to re-integrate their contributions with NCBI-BLAST, acknowledging both the on-going maintenance costs of branches and that potential users strongly preferred to use "real BLAST" whenever possible. Nonetheless for those that were not integrated any desire to integrate was clearly conditional on how integration would occur, turning on the question of whether and how future users would acknowledge the outside contributions, as illustrated in this quote: "the ideal … would be that NCBI [agrees to] write another paper and have them cite this from now on ... that would be the ideal..." The last sentence was inflected in such a way that indicated that such an undertaking might not be ideal for both sides. Thus the authors intentions explanation appears consistent but merely asks a deeper question: why did different authors place different conditions on their desire to integrate? To answer that we turn to our data on incentives.

Motivational conflicts. *The form of motivational rewards that participants seek may mean that conditions under which integration would be mutually beneficial are difficult to achieve. Specifically, the requirements to access academic credit undermine the incentive to integrate, resulting in the maintenance of separate codebases.*

Table 1 combines of our findings regarding integration and incentives: the cases are grouped and shaded according to their code integration status (as in Figure 1), light gray showing those that were integrated and dark gray those that were not. The cells indicate our findings regarding incentives for the three different elements of participation in open collaboration [5].

There is a clear association between integration and incentives: those improvements that were not integrated were motivated in development and revealing by academic credit; those that were integrated were motivated in development and revealing by a mix of motivations, but in integration by a combination of reducing effort and earning revenue through sales of complementary goods.

In essence we found that BLAST innovations from those motivated to improve BLAST by academic reputation are motivated to develop and to reveal, but not to integrate their contributions. Either integration is actively avoided to maintain a separate academic reputation or it is highly conditioned on whether or not publications on which they are authors will receive visibility and citation.

Conversely, other motivations (including use-value, fun, ethics, as well as complementary goods) do not seem to conflict with integration. In fact these motivational rewards seem to be consistent with or actively favor integration, either as a means to reduce on-going maintenance costs or, in the case of complementary goods, to ensure the highest level of comfort using the BLAST improvement that helps sell particular hardware.

We find this explanation to be the strongest: academic reputation as a motivational reward creates conditions that are difficult to satisfy in a mutually beneficial manner, while other motivations do not. This explanation is consistent with all our cases and it was key to the explanations that informants themselves provided.

DISCUSSION

Our result is surprising because reputation has been seen as a key and unproblematic incentive for participation in open software development, as discussed above [e.g., 5,26,37]. Similarly a reputation economy is key to science, with some authors going as far as to describe a separate "republic of

Project	Motivation for …		
	Development	**Revealing**	**Integration**
Integrated Improvements	Revenue (sales) Fun	Revenue (sales) Ethics	Integrated: - Reduce maintenance costs - Increase Revenue (sales)
NCBI BLAST NCBI BLAST+	Academic Credit Use Value Academic Service Provision	Academic Service Provision	N/A
Non-integrated improvements	Academic Credit	Academic Credit	Not integrated: - Counter-motivated or highly conditional due to concerns that integration would undermine academic credit

Table 1: Integration status and motivations for development, revealing and integration (shading refers to Figure 1).

science" where reputation and building on each other's work replaces the dominance of competition for money in other domains [30]. This raises two questions: First, why does reputation appear to be problematic in the production of scientific software, while use-value, fun, learning and even money appear to encourage open collaboration? Second, why does reputation appear problematic for integration in scientific work but not in open source?

We explore possible answers to these questions by examining how different incentives, in content and form, relate to collaboration by using a division of claims approach [5]. The division of claims approach posits that a fundamental issue in collaboration is answering the question of how the overall value resulting from collaboration is distributed amongst those who have contributed. To the extent that that division results in adequate rewards for participants then collaboration will be possible; if that division is not satisfactory collaboration will be threatened.

*Fun and learning***.** A number of the incentives discussed above have rewards that are independent of the overall performance of the system because they are delivered immediately. The fun and learning as a motivation was experienced in the activity itself and existed whether the code was integrated and the overall project improved or not. Thus the rewards of these type of motivations don't have to be divided or transferred to other participants and such motivations are particularly well-suited for open collaboration.

Use-value. The desire of the authors to use their own improvements is an important motivation. This is well known from studies of open source software where developers indicate that they participate to improve the software for their own use [12,37]. In science, use-value derives from one's ability to accomplish (non-software or domain) scientific work that will eventually be published in an academic paper. This type of software has been called "incidental software production" [18] and is clearly powerful for motivating development. It may not, however, be necessary to reveal developments because not doing so may preserve a competitive advantage. In this sense revealing may cause a reduction in the relative value of the use-value to the original author. In science, however, revealing may be a complement to the use-value of development work to the extent that norms of openness and reproducibility (or explicit journal/conference policies) require the full revealing of code in order to publish results based on it (and thus unlock the scientific use value of the software work). Integration, however, appears unlikely to be a complement to use-value (since journals require only revealing and not integration) and so the decision about integration is likely to be based on a separate assessment of on-going costs of maintaining a separate code tree.

Money. Money is a quintessential motivational reward in part because it facilitates a division of claims easily. This is clear in the simple case of for-profit collaboration. If two partners work together to create a valuable system and earn money by selling it they can divide that money in a way that provides sufficient reward to each of them.[4] Revenue from complementary goods sales, as observed above, is more complex but is still a division of claims facilitated by money. In this case the overall value from the improved system is sufficient to generate demand for computers to use the software with. If the software was also sold for money it would be clear that the customer's money was being divided between hardware and software contributors. Money is ideal for valuing and rewarding contributions since it can be divided and distributed easily.

Reputation. Reputation, by contrast, is potentially problematic when viewed through a division of claims framework. Reputation is something that is enacted by

[4] Of course any division is likely a result of power and other strategies but the simple divisibility of money facilitates these strategies.

others through their future actions, especially the extent to which they reveal their regard to others. In the context of BLAST, reputation is an anticipated reward for the development and revealing of software improvements. The improvers anticipate that others will come to value the improvements and therefore regard the improver as a contributor to science. Such regard is useful for career advancement, especially in terms of tenure and respect that leads to jobs, grants and other rewards.

As a motivational reward reputation is not delivered immediately like fun or learning, nor independently like use-value. Rather the value of reputation comes over time and at distance. The fact that reputation is enacted by others means that it is hard to control, just as it is hard to control any actions of others. This is a very strong contrast to financial rewards because once money is received it can be stored, re-directed and divided. In contrast a partner whose rewards come as reputation must work hard to cause the valuable reputation to be redirected or divided, perhaps even to the extent of correcting others each and every time they do not adequately divide their regard appropriately amongst contributors.

The division of reputation is easier to the extent that each individual's contribution is clearly marked in the final product, making a user (or an evaluator) perceive the multiple contributors. When that is the case integration does not diminish reputational reward, and in fact may enhance it by virtue of its inclusion in a more highly-functional product (as with the expected reputation benefit of having one's code included in "real BLAST"). Yet where individual contributions are not visible in the final product, there exists a risk that contributions may be undervalued. Concern over this issue has been seen in sciences with ever-increasing large author lists [8].

More concretely, integrating a piece of code into an existing project risks losing the identity and visibility of that contribution, a concern voiced by an informant that "improvements would be consumed". If the existing project is already firmly associated with its originators, then there is considerable, difficult work to be done to alter the behavior of future users in order to have them recognize the contributions of those whose contributions have been incorporated.

Reputation in open source and science

In open source the issues with reputation and collaboration discussed above seem less problematic than in the academic reputation economy. We see two reasons for this.

The first is that open source projects, and the systems they use, provide insight into the specific, sub-project, contributions of contributors. This can be seen in the use of THANKS files, listing the names of contributors, but even more clearly in the source code management systems where contributions are identified by name, allowing a view of contribution that is disaggregated from the project level. In this way systems like Coderwall can "reach into" integrated products and provide an accounting of contribution in a way that accomplishes a division of claims. Scientific software projects rarely have open repositories that indicate authorship, in BLAST any integrations were performed by core staff under their usernames, not by the author whose code or idea was integrated.

The second is that reputation is more direct in open source than in the academic reputation system. In short reputation in academia is about scientific contribution, whereas reputation in open source is about software work. Scientific contribution, for better or worse, is most often measured through publications and citations, not through other artifacts [17]. This has two implications.

The first implication of indirectness is that contributions have to be substantial enough to warrant a publication that describes them. Since the bar to publication is relatively high, especially in higher quality venues, this seems likely to exert pressure to produce larger contributions, resulting in many lines of code changes. This is a clear contrast to open source where the unit of contribution is the patch and contributions are relatively small [31:324]. Indeed open source norms encourage contributions to be as small as possible.[5] Since smaller contributions are easier to understand they are easier to integrate and so reduce integration costs.

The second implication of indirectness is that contributions must be visible in publications and in citations to those publications. Our informants were clear about this (even as they indicated they were not necessarily pleased about it): they must make their cases through publications and citations. Thus even if a source code repository were to record their integrated contributions if that contribution didn't result in a publication on which they were an author, or if it resulted in shifting citations from one on which they were an author to one on which they were not, then integration would be counter-motivated. Claims from system improvement would not be appropriately divided.

In seems, then, that for collaboration in scientific software work to solve the division of claims problem and motivate integration, contributions must remain visible not only at the software level but at the publication and citation level. This is made even more difficult because papers, unlike source code repositories, are static objects; their author lists do not change over time and thus integrated contributions from those who were not authors would need to occasion a new, joint publication and commitment from the original authors to direct all citations to that new paper.

[5] These norms are summarized, with multiple references, in answer to a question about "code bombs" on StackOverflow: http://programmers.stackexchange.com/ questions/152733/what-is-the-term-for-a-really-big-source-code-commit

POLICY IMPLICATIONS AND FUTURE RESEARCH

Science policy makers are looking for tools and approaches to improve the efficiency and effectiveness of cyber-enabled science, including software development. The success of open collaborations in open source software development and knowledge environments like Wikipedia provide interesting templates, and existing policy suggestions draw heavily on these analogies [19,32,41]. However our work suggests reasons to believe that the specifics of the scientific reputation environment may not adequately motivate academics to integrate their contributions, thus failing to meet the hopes held for open collaboration. Accordingly we argue that policy should address this situation in four ways.

The first approach is to improve returns for integration work. One way is to fund integration directly. For example, funding agencies could provide funding exclusively for integration, bringing together leaders of different projects and requiring the creation of joint papers that not only describe the integration work but provide an appropriate citation target. A related strategy would be to provide funding to groups less as direct sources of software and more as ecosystem "stewards" who are motivated to incorporate and integrate outside contributions [6]. Appropriate models for these include foundations such as Apache or Debian that attend to questions of the software ecosystem but do not directly develop software, aiming to side-step conflicts between core insiders and outsiders. Finally, given such stewards aiming for integration, scientific publishers could exploit the importance of publication and require contributors to go beyond revealing their source code and require them to have integrated their code into common packages before accepting papers for publication (if the claimed contribution is software that others can use). A downside of this approach is that scientists working in this area might see this funding as reducing the money available for their science.

A second approach would be to promote new integration models through architecture. This would imply an explicit preference for an ecosystem with minimal cores and most functionality delivered through separate components, each with their authors clearly identified. Such architectures have been recommended for separate reasons [11,34] but our work suggests that these architectures may be particularly important for collective innovation on scientific software, given the difficulties of direct integration.

A third broad approach would be to attempt to alter the academic reputation economy to reduce the disincentives for integration. One common suggestion is to reward academics for contributions other than journal articles, including data sets [35] and, more rarely, software [17]. Our work suggests that the specific form of non-journal contributions that would be acknowledged would be important in encouraging collaboration. Rather than replacing a static journal article with a record for a static software contribution, encouraging dynamic repositories that facilitate and record small units of contribution may be appropriate. The GitHub model of rapid and open forking may be more appropriate than a centralized model where improvers seek permission to have their improvements added.

A fourth approach, perhaps requiring less alteration to existing practice, would be to provide additional resources for software-contributing scientists to make their case for contribution and to provide disaggregation of contributions. Science policy makers can work with journals, conferences and professional societies to improve the citation of software in publications, specifically encouraging the citation of multiple papers or projects whose code or approaches have been integrated into the packages actually used. This creates a direct visibility for integrated code without needing to alter currents practices too widely. The software itself could provide the appropriate set of citations, keeping track of which code, including integrated code, actually ran and providing a set of citations ready for use in papers.[6] Another resource would be to help software-contributing scientists gather data on the use of their scientific software, through counts of downloads or through instrumentation of the code or distributions [28,42]. Ideally such data would link to publications and expose integrated contributions and thereby allow those making software contributions to demonstrate their scientific impact even when their code has been integrated, much as services like Coderwall and Ohloh re-aggregate contributions in open source.

Finally, the work in this paper also has implications beyond science. As CSCW systems continue to facilitate the growth of novel collaborations at scale, designers will increasingly wrestle with the challenges of working with alternative motivational rewards and their varying socio-material characteristics. We contribute to this effort by showing that different elements of open collaboration—development, revealing and integration—should have their motivations analyzed separately. In particular our results should encourage researchers and designers to consider how their reputation systems realize recognition of contribution at a sub-project level and thus support an appropriate sharing of the spoils of open collaboration.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-1111750, SMA-1064209, and OCI-0943168.

[6] The R statistics community provides part of this solution. Each package is encouraged to provide an implementation of the `citation()` command. This returns a citation appropriate for a citation manager and use in papers. Currently, however, this command does not return citations for dependencies of that package; such an extension is possible. Similarly it ought to be possible to analyze a set of analysis scripts and return appropriate citations.

REFERENCES

1. Alexander, J. *Software Sustainability through Investment*. 2009.
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. Basic local alignment search tool. *J Mol Biol 215*, 3 (1990), -410.
3. Atkins, D. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure. 2003. http://www.nsf.gov/od/oci/reports/toc.jsp.
4. Baldwin, C.Y. and Clark, K.B. *Design Rules: The Power of Modularity*. Harvard Business School Press, 2001.
5. Baldwin, C.Y. and Clark, K.B. The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Management Science 52*, 7 (2006), 1116–1127.
6. Berente, N., Howison, J., and King, J.L. *Report on Workshop on "Managing Cyberinfrastructure Centers"*. University of Georgia.
7. Bietz, M.J., Ferro, T., and Lee, C.P. Sustaining the development of cyberinfrastructure: an organization adapting to change. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ACM (2012), 901–910.
8. Birnholtz, J. When Authorship Isn't Enough: Lessons from CERN on the Implications of Formal and Informal Credit Attribution Mechanisms in Collaborative Research. *Journal of Electronic Publishing 11*, 1 (2008).
9. Bower, G.C., Edwards, P.N., Jackson, S.J., and Knobel, C.P. The Long Now of Cyberinfrastructure. In *World Wide Research: Reshaping the Sciences and Humanities*. MIT Press, Cambridge, MA, 2010.
10. Camacho, C., Coulouris, G., Avagyan, V., et al. BLAST+: architecture and applications. *BMC bioinformatics 10*, 1 (2009).
11. Cataldo, M., Herbsleb, J.D., and Carley, K.M. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08)*, (2008).
12. Crowston, K., Wei, K., Howison, J., and Wiggins, A. Free (Libre) Open Source Software Development: What We Know and What We Do Not Know. *ACM Computing Surveys 44*, 2 (2012), open–source–software–development–what–we–know–and–what–we–do–not–know.
13. Dabbish, L., Stewart, C., Tsay, J., and Herbsleb, J.D. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *CSCW*, (2011).
14. Faniel, I. *Unrealized Potential: The Socio-Technical Challenges of a Large Scale Cyberinfrastructure Initiative*. 2009.
15. Gambardella, A. and Hall, B.H. Proprietary versus public domain licensing of software and research products. *Research Policy 35*, 6 (2006), -892.
16. Ghosh, R.A., Robles, G., and Glott, R. *Free/Libre and Open Source Software: Survey and Study FLOSS*. 2002.
17. Hafer, L. and Kirkpatrick, A.E. Assessing open source software as a scholarly contribution. *Commun. ACM 52*, 12 (2009), 126–129.
18. Howison, J. and Herbsleb, J.D. Scientific software production and collaboration. *Computer Supported Collaborative Work (CSCW 2011)*, (2011).
19. Ince, D.C., Hatton, L., and Graham-Cumming, J. The case for open computer programs. *Nature 482*, 7386 (2012), 485–488.
20. Knepper, R. and Repasky, D. *Open Source Writ Large: Advantages of a Foundation Community Model for Cyberinfrastructure*. 2009.
21. Kraut, R.E. and Resnick, P. *Building Successful Online Communities: Evidence-Based Social Design*. The MIT Press, 2012.
22. Lakhani, K. and Wolf, R.G. *Why hackers do what they do: Understanding motivation efforts in Free/F/OSS projects*. 2003.
23. Lawrence, K.A. Walking the Tightrope: The Balancing Acts of a Large e-Research Project. *Computer Supported Cooperative Work 15*, 4 (2006), 411.
24. Lee, C.P., Bietz, M.J., Derthick, K., and Paine, D. A Sociotechnical Exploration of Infrastructural Middleware Development. (2012).
25. Lee, C.P., Dourish, P., and Mark, G. The human infrastructure of cyberinfrastructure. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, ACM (2006), 483–492.
26. Lerner, J. and Tirole, J. Some simple economics of Open Source. *Journal of Industrial Economics 52*, 2 (2002), -234.
27. MacCormack, A., Rusnak, J., and Baldwin, C.Y. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science 52*, 7 (2006), 1030.
28. McConahy, A., Eisenbraun, B., Howison, J., Herbsleb, J.D., and Sliz, P. Techniques for Monitoring Runtime Architectures of Socio-technical Ecosystems. *Workshop on Data-Intensive Collaboration in Science and Engineering (CSCW 2012)*, (2012).
29. Merton, R.K. and Sztompka, P. *On Social Structure and Science*. University of Chicago Press, 1996.
30. Merton, R.K. The Matthew effect in science, II: Cumulative advantage and the symbolism of intellectual property. *Isis 79*, 4 (1988), 623.
31. Mockus, A., Fielding, R.T., and Herbsleb, J.D. Two Case Studies Of Open Source Software Development: Apache And Mozilla. *ACM Transactions on Software Engineering and Methodology 11*, 3 (2002), -346.
32. Morin, A., Urban, J., Adams, P.D., et al. Shining Light into Black Boxes. *Science 336*, 6078 (2012), 159–160.

33. Olson, G.M., Zimmerman, A., Bos, N., and Wulf, W. *Scientific Collaboration on the Internet*. 2008.
34. Parnas, D.L., Clements, P.C., and Weiss, D.M. The modular structure of complex systems. *IEEE Transactions on Software Engineering 11*, 3 (1981), 266.
35. Piwowar, H.A., Vision, T.J., and Whitlock, M.C. Data archiving is a good investment. *Nature 473*, 7347 (2011), 285–285.
36. Raymond, E.S. The Cathedral and the Bazaar. *First Monday 3*, 3 (1998).
37. Roberts, J.A., Hann, I.-H., and Slaughter, S.A. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science 52*, 7 (2006), 999.
38. Science Watch. Twenty Years of Citation Superstars. *Science Watch 14*, 5 (2003).
39. Stallman, R. and others. The GNU manifesto. *Dr. Dobb's Journal of Software Tools 10*, 3 (1985), 30–35.
40. Stewart, C.A., Almes, G.T., and Wheeler, B.C., eds. NSF Cyberinfrastructure Software Sustainability and Reusability Workshop Report. 2010. http://hdl.handle.net/2022/6701.
41. Stodden, V., Donoho,, D., Fomel, S., et al. Reproducible Research. *Computing in Science and Engineering 12*, 2010, 8–13.
42. Thain, D., Tannenbaum, T., and Livny, M. How to measure a large open-source distributed system. *Concurrency and Computation: Practice and Experience 18*, 15 (2006), John Wiley and Sons Ltd.– 2019.
43. Vertesi, J. and Dourish, P. The Value of Data: Considering the Context of Production in Data Economies. *CSCW 2011*, (2011).
44. Vouzis, P.D. and Sahinidis, N.V. GPU-BLAST: Using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, (2010).
45. Wagstrom, P., Herbsleb, J.D., Kraut, R.E., and Mockus, A. The Impact of Commercial Organizations on Volunteer Participation in an Online Community. *Presentation at the OCIS Division, Academy of Management Conference*, (2010).