Community Code Engagements: Summer of Code & Hackathons for Community Building in Scientific Software

Erik H. Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, James D. Herbsleb
Institute for Software Research
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{etrainer, cchaihir, arunkaly, jdh}@cs.cmu.edu

ABSTRACT
Community code engagements — short-term, intensive software development events — are used by some scientific communities to create new software features and promote community building. But there is as yet little empirical support for their effectiveness. This paper presents a qualitative study of two types of community code engagements: Google Summer of Code (GSoC) and hackathons. We investigated the range of outcomes these engagements produce and the underlying practices that lead to these outcomes. In GSoC, the vision and experience of core members of the community influence project selection, and the intensive mentoring process facilitates creation of strong ties. Most GSoC projects result in stable features. The agenda setting phase of hackathons reveals high priority issues perceived by the community. Social events among the relatively large numbers of participants over brief engagements tend to create weak ties. Most hackathons result in prototypes rather than finished tools. We discuss themes and tradeoffs that suggest directions for future empirical work around designing community code engagements.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation (e.g., HCI)**]: Group and Organization Interfaces – *computer supported cooperative work, organizational design.*

Keywords
Community code engagements, Google Summer of Code (GSoC), hackathons, scientific software.

1. INTRODUCTION
How do you go from a small number of people with a common interest to a full-fledged community? The active body of research on this problem (e.g., [17, 30, 31]) is a testament to the role of community building in collaborative work practices.

Active communities are essential to the sustainability of software. Without a community around the code distribution, key issues of the software's future may not be addressed, e.g.: in 4 years' time, will the software still be available? Will it work? Will there be pool of participants with the right set of technical skills who can respond to bug reports and feature requests? Successful communities find ways to get code contributions from their members and to incorporate successive generations of newcomers after the original developers leave [30].

There is an additional twist for software that scientists write. Although scientists are directly funded to produce new knowledge, they spend significant time searching for, using, and developing software that enables those results. The sustainability of scientific software – the ability to maintain the software in a state where scientists can understand, replicate, and extend prior reported results that depend on that software – has sometimes been an afterthought because scientists are rewarded for the publications they write, not the software they create and support [25, 26]. This software, however, is a critical link in the chain of evidence establishing new scientific knowledge, and thus other scientists need to be able to run this software in order to understand and replicate this new knowledge, and apply it to new problems.

A few scientific communities in the life sciences have begun experimenting with short-term focused community engagements, such as Google Summer of Code (GSoC) and hackathons. Although there is reason to believe from previous research on online communities (e.g., [30]) that these engagements may enhance the sustainability of scientific software, there is as yet little empirical support. Moreover, evidence about when various types of engagements are likely to succeed is scant.

In this paper, we aim to understand the range of outcomes these engagements produce and the underlying practices that lead to these outcomes. We hope to highlight concrete engagement design issues that community leaders and funding agencies might consider in order to optimize the outcomes they desire.

2. BACKGROUND
2.1 Sustainability of Scientific Software
Software is of vital importance to science. The role of software in data analysis, simulation, and visualization is widely acknowledged [9, 27, 38]. A 2005 NSF Workshop Report [5] clarified the importance of software in cyberinfrastructure, which is the "infrastructure based upon distributed computer, information, and communication technology"[2]. Much scientific software, however, is not infrastructural. For instance, there are many "workbench" applications for end user scientists (e.g., Dan Gezelter's directory [34] lists almost 500 programs). This list does not even include the myriad scripts and data conversion utilities scientists write to translate data into intermediate forms required by tools in the later stages of workflows [26]. Although NSF sponsored workshops have repeatedly called attention to cyberinfrastructure maintenance [5], there is less clarity into how scientific software more generally can be sustained over time,

GROUP '14, November 9–12, 2014, Sanibel Island, FL, USA.

http://dx.doi.org/10.1145/2660398.2660420

even though it is a crucial part of scientific research, development, and delivery [44].

Informal evidence suggests that scientific software is increasingly a key problem of interest to individual researchers and research institutions. For example, the First Workshop on Sustainable Software for Science [51] was recently held collocated with an annual conference on High Performance Computing. A simple head count revealed that one third of the workshop participants only attended the conference for the workshop. As further evidence, the Water Science Software Institute has developed a model for software development specifically aimed to support the maintenance of scientific software [6].

Scientific software exists in a variety of states from "as persistent as the next grant supporting maintenance" to "supported by a very small community of volunteers" [44]. But if that software has gained widespread use outside of the lab and served a valuable role in assisting other scientists in making new discoveries, it should be able to be refined and extended for use by other scientists who can use it to produce new knowledge.

2.2 The Promise of Open-Source

In discussions of software sustainability, the open-source software model is invariably held up as a promising approach. For instance, position papers from the 2009 NSF funded workshop on "Cyberinfrastructure Software Sustainability and Reusability" [44] led to the report's recommendation that cyberinfrastructure software should be released under an open-source license.

Directly applying the open-source model to scientific software development, however, neglects crucial differences between open-source scientific software and open-source software in general. The primary difference is in the incentive structure for contribution [25, 26]. For open-source developers, reputation in the open-source community is a primary motivation, where the number of "followers" a developer has is a symbol of social status [13]. Scientists who write software, in contrast, operate in a "reputation economy of science" that rewards software production only indirectly through publication [26].

Because scientists need tools for their own work, however, there is built-in motivation to put time and effort into developing them. Although scientist developers are reluctant to build software they themselves do not need, scientists are, under certain conditions, willing to undertake extra work needed to turn their personal tools into a community resource [48]. For example, if scientists understand users' needs well enough, they will be more inclined to devote effort to building features that meet those needs.

Newcomers, however, will likely face multiple barriers to entry: for example, installing the tools, learning technical aspects of the codebase, and learning social conventions, such as where to post questions or issues and how to contribute code. As we review below, research suggests that a healthy community can help lower the barriers to participation and contribution. The reverse is also true: lowering the barriers helps grow a healthy community.

2.3 Online Communities

At a recent workshop on the sustainability of scientific software [51], over half of the 57 papers accepted mentioned *community* as a crucial ingredient in the recipe for success of scientific software sustainability. A community is a group of people who share a common interest, purpose, or goal. Practitioners who learn from each other to develop themselves personally and professionally (e.g., a less experienced scientist developer works with a more experienced developer to develop features of increasing difficulty) constitute a community of practice [33], whereas people who share information with others but who are not necessarily practitioners themselves (e.g., an end user scientist answers a question about the tool's installation procedure on the mailing list) constitute a community of interest [24]. In a community of practice, learning is always situated in practice. Prior studies of open-source software communities have used *situated learning* to describe the socialization and sustained participation of newcomers [17]. This suggests that community of practice is the type of community important for the sustainability of scientific software.

The literature on communities suggests that in order to be successful, communities must address two primary challenges: receiving contributions and attracting newcomers.

2.3.1 Contributions

Communities need contributions from participants ([30], p. 2-5). In software development, contributions might comprise a working base of source code that serves people significantly better than the competition. For example, BLAST, the widely used open-source tool for comparing biological sequence information [1] achieved such early success because it was so much faster than previous algorithms available at the time. Assuming a newcomer is at least interested in modifying or contributing to the software, the barrier to entry must be low to modify and extend the existing contributions to meet their own needs. Because most contributors to open-source software projects leave after their personal needs are met [41], it is important to attract enough newcomers in order to find those who will continue to contribute and act as stewards of the code base.

2.3.2 Attracting Newcomers

To survive over the long-term, **communities must find ways to attract new generations of members** to replace the ones who leave ([30], p. 179). For example, in recent years leaders of the online encyclopedia Wikipedia have expressed discontent over the fact that the growth rate of new contributors does not compensate for the number of experienced article editors who drop out [3]. Incorporating newcomers can be challenging for three reasons. First, newcomers have not yet developed the same commitment to the group as older members. Second, newcomers will not know how to contribute as effectively as older members. Third, they will not be aware of the norms and objectives guiding the group. In open-source software projects, learning how to contribute to the codebase is often informal and undocumented, placing the onus on the would-be developer instead of the group [15, 31].

2.3.3 Designing Communities

Research on online communities also tells us that **the features of online communities can be designed and managed to achieve the goals that their members desire** ([30], p. 6). For example, designers of communities can encourage new participants and prepare them to make contributions by providing them with welcoming activities, safe spaces for exploration, and formal training opportunities. The size of the community can make a difference as well; more participants can potentially contribute more content. Furthermore, tasks taken on by members can be independent or interdependent, and they can be embedded in social experiences. Rewarding or sanctioning users in response to the actions they take can motivate or demotivate them to make additional contributions.

The fact that communities are so important to software sustainability, and that they are amenable to management and design, suggests that we can improve the sustainability of

scientific software by intervening to improve the state of its community.

2.4 Community Code Engagements

Prior studies suggest that there is reason to believe that *community code engagements* – short term, intensive, software development events – may be an effective way to put scientific software projects on sustainable trajectories [6, 47]. Open-source scientific software communities, particularly those in the life sciences, have started employing two types of engagements: Google Summer of Code (GSoC) and hackathons.

2.4.1 Google Summer of Code (GSoC)

Google Summer of Code (GSoC) is an annual program that pays university students stipends to develop features for various open-source software projects. It aims to familiarize students with open-source software development and enable projects to more easily identify and bring in new contributors [20].

The GSoC application requires mentoring organizations, individuals or organizations running an active open-source software development project, to provide details about the organization, previous participation in GSoC, communication methods, plans for project management and participant engagement, and a URL to their project "idea list." The idea list contains descriptions of potential projects, potential mentors, and required skills for students. Mentors are community members who volunteer to provide students technical and social support.

If Google accepts an organization's application, students then submit proposals to the organization describing features they wish to develop. Students can expand on projects in the idea lists, or propose new projects. The organization reviews and ranks student proposals, assigns project mentors and students, and requests project slots from Google.

Google allocates each accepted organization a certain number of project slots, and the organization assigns students and mentors to those slots. Before coding begins, mentors introduce students to the community's culture and practices, select communication channels to use, and decide on the frequency and form of status reports. Throughout the coding period, mentors oversee their students, providing help and guidance. Coding lasts three months, after which students submit their final projects to Google. Google pays students once at the beginning of the coding period, and at the middle and end, provided that they pass mid-term and final evaluations.

2.4.2 Hackathons

Hackathons are events typically lasting two to seven days where people meet face to face and collaborate intensively on software. Hackathon experience reports tend to place emphasis on the number of new tools, prototype functionality, and lines of code produced (e.g., [28, 29, 32]). Because the number of participants at hackathons can be high (having at least 30 participants is common, according to papers in our review), these events have the potential to significantly contribute to a codebase. Hackathons also provide opportunities for networking and relationship building, which may facilitate incorporating new generations of newcomers.

Hackathon organizers typically solicit participants, who may be developers of scientific software, or end user scientists, through private invitation or open calls on mailing lists. Before the event starts, organizers may hold "bootcamps," short tutorials designed to help developers new to a toolkit get acquainted with the codebase and the functionality it offers. When the hackathon starts, participants make presentations on topics they would like to work on. Based on these presentations, participants collectively consolidate and prioritize development tasks and then form sub-groups of about five people based on common interests and project affiliations. For the duration of the hackathon, sub-groups work collocated in a large room on their respective tasks and hold short meetings at the beginning of every day in which group members give status updates on their work. At the end of each day, there is often a dinner, reception, or short excursion where participants eat, drink, and socialize with each other.

We note that hackathons have been widely applied in areas outside of science, such as Yahoo! Open Hack Day, where developers create applications using Yahoo! APIs. For a fairly comprehensive list of the varieties of hackathons, see [22]. In this paper, we focus on hackathons used in the sciences.

3. METHOD

To understand the range of outcomes GSoC and hackathons produce and the underlying practices that lead to these outcomes, we performed a multiple case study of 22 GSoC projects within 6 different scientific software projects across 3 different domains (Table 1). Searching the project lists on the GSoC home page [20] for scientific software, we found that the proportion of bioinformatics GSoC projects is high compared with other scientific domains, which is reflected in our sample. Another criterion was to pick projects with a track record of participation in GSoC so that we could tell if participants' project activities carried over into subsequent years. Our last criterion was to pick active projects, which would indicate that the software serves a current scientific need.

We found very active projects in Biopython, Cytoscape, Wikipathways, CGAL, Bioconductor, and VTK. Each project has consistent development activity, a well-used wiki for documentation and discussion and publicly available mailing lists. Biopython is a set of Python libraries for biological computation [7]. Cytoscape is a software platform for visualizing molecular interaction networks [42]. Wikipathways is a wiki for contributing and maintaining content related to biological pathways [36]. The Computational Geometry Algorithms Library (CGAL) is a software library that provides access to efficient algorithms for computational geometry [16]. Bioconductor provides R packages for analysis and comprehension of genomic data [19]. Visualization Toolkit (VTK) is a software tool for 3D computer graphics, image processing, and visualization [39]. For each project in our sample, we looked at the software community's website and mailing list archives in order to identify the mentors and students and discussions of project ideas and feedback.

3.1 Data Collection

To find instances of hackathons, we conducted a review of the research literature on hackathons using Google Scholar and regular web searches. If we found a publication describing a particular hackathon, we triangulated information about the number and types of participants, activities, objectives, and outcomes described in the publication with information from any planning documents (e.g., agendas, calls for participation, write-ups of work accomplished) linked to on the engagement's web page. This process resulted in a list of 42 hackathons.

In addition to collecting archival data, we conducted semi-structured interviews with 38 scientific software developers.

Table 1. Scientific software projects and number of GSoC projects in our sample.

Scientific Software Name	Scientific Domain	Number of GSoC Projects
Biopython	Bioinformatics	8
Cytoscape	Scientific Visualization	3
Wikipathways	Bioinformatics	3
CGAL	Mathematics	4
Bioconductor	Bioinformatics	2
VTK	Scientific Visualization	2

We also targeted scientist developers who attended hackathons, some of whom attended the same hackathon, and some of whom attended different ones, in order to see variations in their perceptions of the benefits, challenges, and outcomes. We conducted interviews using either Google Hangout or Skype. Each interview lasted 45 minutes on average. If participants could not commit to an interview, we e-mailed them questions from our interview protocol. Twenty-four people participated in GSoC; 15 participated as students, 5 participated as mentors, and 4 participated as both students and mentors. Six of our 38 participants participated in hackathons. Two participated in the 2012 and 2013 OpenBio Codefest Hackathons [8], 2 participated in the 2013 RMassBank Hackathon, 1 participated in the 2012 NESCent Phylotastic Hackathon [35], and 1 participated in the 2012 SWAT4LS Hackathon [46]. Three people participated in both GSoC and hackathons.

3.2 Data Analysis

We applied a grounded approach [10] to analyze our interview data, and started analysis while the data was being collected. All interviews were recorded, transcribed, and prepared for analysis in the Dedoose qualitative data analysis software [43]. We began analysis by conducting open coding on statements about practices and outcomes associated with GSoC and hackathons. Our research group met weekly to define and discuss codes, compare instances of coded excerpts to previously examined examples, and unify them where there was commonality. Moreover, we triangulated statements in the interviews, such as references to mentors, students, and project status with the archival data we collected on GSoC and hackathons. In the next phase of analysis, we wrote, shared, and discussed descriptive memos about the role of GSoC and hackathons.

4. RESULTS

We found that each engagement has periods where participants define development targets and interact with other community members. These phases result in similar outcomes, but in different ways. We describe these processes and their outcomes in the sections below.

4.1 GSoC

A GSoC project is often structured such that the student works on a particular module that is independent of the rest of the codebase. For example, in the Biopython project, we observed that students created a separate GitHub branch for their code. In the Bioconductor project, students worked on a standalone package that contained all the code necessary to implement one piece of functionality. GSoC projects proceed without concerns that the development will break functionality in the existing code distribution.

4.1.1 Defining Development Targets

Among other things, the GSoC application requires each mentoring organization to submit an idea list. The idea list is meant to introduce contributors to the needs of the project and to provide inspiration to would-be students. It is framed as a starting point for student applications, but we observed a range of projects, from those expanding on a proposed idea, to those not mentioning items in the idea list at all.

We investigated where the ideas in the idea list came from, how those ideas were prioritized, and how student projects were established.

4.1.1.1 Creating the Idea List

The decision for participating in GSoC usually emerged from discussion among core community members, those who were actively involved in project management, or who contributed the majority of the code. Because of their experience and expertise, they were sometimes quite sure about what ideas should be implemented as part of GSoC:

"*...we did it in a controlled kind of a way, that we discussed it internally and amongst the [group name], which are, say a group of power users that...And from there we identified projects that would be useful to pursue." (P31)*

One member of the core team informed us that most projects will have a to-do list or wish list that community members would like people to work on. GSoC provides an opportunity to get these ideas rolling. As he said:

"…there are some ideas on the Wiki, and also in the bug tracker, and sometimes they just sort of get passed around on the project mailing lists. But the developers, in general, have more ideas than time on their hands…" (P4)

Those initial ideas generated by the core team members would be placed on a webpage designed for showing existing ideas and providing details about them[1]. Core team members also encouraged community members and potential student applicants to contribute their ideas, or build on the existing ideas presented on the webpage.

We found that sometimes the core team member would send the call for ideas to the community mailing lists. Biopython also used a Wiki page to facilitate contribution of ideas [4]. The ideas reflected the visions and needs of individuals. Most ideas came from their own research, problems they faced, lack of certain features in the software, new things they wanted to explore, or common needs that were discussed among members.

4.1.1.2 Prioritizing Ideas and Establishing Projects

Since only a few proposals will be accepted, the task of ranking and prioritizing the proposals becomes important. The criteria to prioritize these ideas depend on the views of the core team members. Typically, ideas must be generally useful, practical, and able to be completed within the GSoC time frame.

The availability of mentors and the potential of student applicants were also taken into account. We found that, in general, the match between mentors and students had a big influence on which ideas were prioritized over others. Ideas proposed by experienced

[1] For an example of a project idea page, see https://www.cgal.org/project_ideas.html

mentors that attracted students who showed a strong promise to complete the work and potential to continue to make contributions after GSoC, were assigned high priority.

The number of ideas that would be selected depended on the slots allocated by Google for the mentoring organization. Once the mentoring organization selected ideas based on the number of project slots, the mentors and the students could define the directions of the project by themselves. The code of the student projects was usually visible and the community members were welcome to give them feedback.

4.1.2 Relationship Building: Interactions with Mentors Create Strong Ties; Sharing Updates Creates Weak Ties

GSoC students build relationships with mentors as well as other members of the community. We use the term *strong tie* to refer to relationships that involve frequent communication, which may lead to bonding. We use *weak tie* refer to relationships that involve occasional communication, which may lead to exposure to novel information [21].

4.1.2.1 Strong Ties

One of the recurring themes among GSoC students was the strength of the mentor-student relationship. Almost all students we interviewed expressed a deep sense of bonding with their mentor. Students begin their interaction with mentors during the proposal phase of the project, usually via private emails or video chats. There are usually very few face-to-face interactions and yet we found that over time, students developed a strong relationship with their mentor. One student expressed this as follows:

"Communication with my mentor was great, initially it was email communication and in July I attended the Bioconductor's conference that they have every year. After the conference we kind of transitioned into this Google hangouts weekly. And now he is actually going to serve on my thesis committee." (P48)

We found that GSoC students get direct feedback from their mentors on a frequent basis, which supplements our previous findings from a quantitative analysis of communications between students and mentors in Biopython GSoC projects [47]. As such, students are able to focus more on getting work done instead of collating feedback from different members of the community, which is typical of open-source software communities [15].

Prior to GSoC, a number of students had only limited software development experience (e.g., P5, P27) and others had none at all (e.g., P34). Mentors help these students develop skills around unit testing, APIs, GitHub, object-oriented programming, and reading others' code.

We found that mentoring is often far-ranging, not just about developing source-code. For example, mentors give GSoC students advice on choosing career paths:

"A lot of the students that we get are probably in the middle of grad school, which is the time when you're most like I don't know what's going on. So people have asked me about career paths and those sort of questions. So definitely I've tried to help people in that way too. And because I had all those same experiences. So it's like I can at least relate to them." (P1)

We also found evidence that students are likely to stay in touch with their mentors after GSoC ends. Students across different projects in our sample (e.g., P33, P38, P40, P43) mentioned that they were only in touch with their mentors, suggesting that they did not continue to communicate with other project members.

4.1.2.2 Weak Ties

The way students interact with other members of the community during the course of their project depends on how their progress is shared. We found two distinct ways in which this was done: private communication and announcement of updates on mailing lists. In the case of private communication, students typically maintain their code in a private repository shared only with mentors. In such cases, sharing updates on their progress is sometimes implicit, which involves submitting changes to the shared repository without explicitly updating their progress via emails, as one participant from Bioconductor described:

"Most of the updates went to either <mentor>(P26) or <a core member>, basically everything was done through SVN. So, at any point they could come and check what changes I have made." (P48)

Upon probing further about receiving any requests or feedback from other members, the participant (P48) mentioned that there was not much engagement on the mailing list except for a short period after his package was included in a Bioconductor release.

On the other hand, in Biopython, we found that in addition to private communication between students and mentors, it was mandatory for students to share their progress on mailing lists and blogs. This increased visibility and the potential to garner feedback and different points of view from other members. Making the code available on a public repository like GitHub during the development phase was also another way of increasing community engagement. One of the Biopython participants who was also a GSoC mentor, explained this process as follows:

"Some discussion actually happens on Github itself through the commenting on there. And we also wanted our students to do a weekly blog post. So every week they would try and summarize what they'd been doing. And then post the link on the mailing list, and post it up on our blog as well." (P7)

In sum, we observed that establishing a formal protocol of making GSoC project updates available to the community allowed students to form weak ties with other community members.

4.1.3 Project Outcomes

4.1.3.1 Actively Used Software

The primary objective of GSoC projects is to produce code. However, it is unclear how this code is used and maintained few years after a GSoC project ends. Fourteen of the 22 GSoC students responded that their code written as part of GSoC was still being used and maintained. The most common way to gauge the usefulness of the code was by identifying if it was included in the official release. We saw instances, however, where the project resulted in scientific publications (e.g. P46) and in one case the software itself was being cited in other scientific publications (e.g. P42). All 14 participants who expressed positively that their code was being used were also aware of how their code was currently being maintained, and in some cases the participants continued to maintain the code after GSoC.

Some of the reasons why the participants thought their code was not useful or were not aware if their code was being maintained were change in field of work and re-implementing their code using a different technology:

"Once GSOC finished I had almost zero contact with the project. This was also because my line of work after graduating did not fall in line with the project (am doing clinical instead of research-related work). A few months after GSOC, there was an email thread going on regarding how to best deploy the work done after

my mentor made several improvements on it, but by then my interest on the project had faded so I did not look much into it." (P43)

4.1.3.2 Student Retention

One of the motivations of this process of facilitating ties between newcomers and other members of the community is to retain their engagement and future contributions. We found that the possibility of students continuing to be involved in the community was also one of the criteria in selecting students. The following quote summarizes the rationale behind student selection:

"Sometimes people just work for the summer and that's it. And then sometimes they continue. And we really try to keep this in mind when people are selected. You really want someone that's going to keep going and keep contributing to the community." (P1)

While some students ramped down their involvement post GSoC, other students went on to become mentors, active contributors and users. Among the 22 GSoC projects we studied we found that 18% of students went on to become mentors later, a reasonably strong indication of retaining new comers in a community. One participant who has been actively involved in the community for several years since being a student said:

"I am still involved. I co-mentored and mentored some Cytoscape projects from 2009-2011, joined a couple of retreats, published a book on Cytoscape, and some of my other research are still closely related to Cytoscape though I am not part of the core development." (P46)

There were few participants who mentioned their involvement with the community stopped after their GSoC project ended, although we found that they had still retained some of their ties that they formed during course of their project:

"No contact once GSOC ended. Shortly after the GSOC though, a professor in close relationship to the project came to visit Singapore and I met him plus one other GSOC student and had a chit-chat together. It was nice, but no activity after that." (P43)

4.2 Hackathons

Whereas the GSoC format works well for independent work, the hackathon format seems more designed for interdependent work**.** Participants recalled that hackathon organizers purposefully select objectives that require intensive coordination and collaboration because it is maybe the only time they will be able to do this kind of work (e.g., P7, P20). For example the goals of the 2008 DBCLS Biohackathon [29], and the 2006 NESCent Phyloinformatics Hackathon [32] were to increase the level of interoperability and standardization of bioinformatics databases, services, and tools. By their nature, standardization and interoperability require cooperation from independent tool builders, database providers, and standards bodies. These stakeholders, however, are often geographically dispersed across multiple time zones, and live interactions such as conference calls would have to be scheduled outside normal office hours.

Hackathons last up to several days in length (the average length in our sample was 3.3 days), which provides a long but limited span of uninterrupted time to work. A short period of issue prioritization and development target identification precedes intensive and focused coding sessions.

4.2.1 Defining Development Targets

Hackathons begin with an agenda setting phase in a large shared room, where the output is a list of tasks that participants will work on for the duration of the event. Each participant who wants to raise an issue gets the opportunity to speak before the group. One participant summarized the process as follows:

"…the very first day, we had a few presentations to kind of give the scope of the hackathon, what we wanted to accomplish… [presenters] would say, 'I think that would further our goals would be to develop this. And I'm looking for collaborators to work on this with me.'" (P30)

After hearing from presenters, attendees collectively consolidate the total number of issues, translate them into development targets, and prioritize them into two categories: (1) issues that can be directly addressed at the event and (2) issues that cannot be addressed in the time allotted. For example, a hackathon planning document from the 2006 NESCent Phyloinformatics Hackathon [32] indicates that 6 out of the original 19 "use cases" proposed by participants were ultimately selected for development targets. The following comment on the "Population Analysis" use case seems to indicate that, while not selected for the hackathon, it is still of value to the community:

"This [use case] is outside the scope of the current hackathon but will be addressed in a future one --Tjv *10:26, 21 October 2006 (EDT)"* [49]

Through this process, participants can effectively identify and prioritize the common challenges facing the community. When otherwise geographically and temporally dispersed from their closest collaborators, scientists may wonder whether the issues they struggle with matter to others in the same way. The hackathon answers this question with clarity. Once participants sort these issues out, they break into sub-groups and work intensively on the development targets they identified.

4.2.2 Relationship Building: Sub-group Work Creates Strong Ties; Social Events Create Weak Ties

Hackathon participants gather together from all over the world. While they may know the names of their colleagues and something about the kinds and level of their activities from project mailing lists, issue trackers, and source-code repositories, they have fewer opportunities to get to know them. We found that working in sub-groups creates few strong ties among members, and that social events create weak ties among a relatively larger number of people.

4.2.2.1 Strong Ties

The number of people who attend the hackathon can influence the degree to which participants build upon their current relationships, because there is a limit to the time they can interact outside of sub-groups. As one participant described:

"At larger hackathons, like the BioHackathon series which has had about 80 participants, inevitably as with a similarly sized workshop/conference I am unlikely to get to speak to everyone. I would hope at least to remember key people giving talks or representing a sub group in progress meetings/wrap-up sessions." (P7)

We found that sub-group work creates strong ties among group members. Participants spend the majority of their time working together in sub-groups. During this time, debugging code, explaining thought processes, asking for feedback, and offering tips and suggestions lead to frequent interactions. Working together allows participants to develop deeper relationships with individuals within the group:

"I find you get to know the people better if you actually work together with them and see how they react to problems along the

way. I really need to know this about a person to be able to work with them well." (P36)

4.2.2.2 Weak Ties

Outside of sub-group interactions, social events provide social networking opportunities with the larger group. Of the hackathons in our sample, 79% (33/42) included short social events, such as coffee breaks, group dinners, and group excursions to nearby landmarks of interest. These events provide participants with opportunities to chat informally with other hackathon attendees, creating weak ties that expose them to information about:

Interesting tool developments or research in one's area. Several of the developers we talked with spent a lot of time discussing technologies and tools that might benefit their research. We interviewed a developer (P20) who introduced an end-user (P36) to an online community that distributes R packages for conducting analyses of genomic data. The end-user is now an active reader of that community's forums and user of the packages that the developer writes. They have also co- authored articles, worked together at subsequent hackathons, and provided feedback on each other's work.

Suggestions for future conferences of interest. Participants reported hearing about conferences and people working on related things through researchers that they met at hackathons (P7, P23, P36). In some cases, they led participants to whole software communities. One of our participants, for example, reported that a developer she met at a hackathon introduced her to the Bioconductor community (P36).

News of job vacancies. The participants we spoke with also told us that hackathons are a useful way to hear about job openings that open up, both academic positions and positions in industry (e.g., P7).

Invitations to tutor or speak at other conferences. One participant told us that he just got back from contributing to a Python part of a workshop tutorial on "keystone skills for bioinformatics." (P7) Another scientist who he had first met at a hackathon invited him.

4.2.3 Project Outcomes: Limited Time Results in Unfinished but Promising Outcomes

Participants we spoke with indicated that it is rare that "finished" software is produced as a result of the hackathon. As one participant recalled, *"we did a lot of programming in that meeting but had a long way to go still."* (P36). We found that participants often attributed the reason for this to the limited time allotted for the hackathon:

"Of course we came across more problems than anticipated and didn't get as much done as we wanted…the task was much bigger than 1 day." (P36)

Instead, software outcomes ranged from *"not anything to write home about" (P31)* to *"discarded—serving as inspiration for a second attempt"* to something *"useful enough that the authors can polish it afterward." (P7)*

We found that the most common software products were:

4.2.3.1 Integration of Existing Tools, Web Services, and Databases

Outcomes improved the interoperability of existing tools with other tools, services, and databases. For example, at the NESCent 2006 hackathon [32], developers of the Bio* toolkits (i.e., BioPerl, BioJava, and Biopython), expanded their coverage of data types and analyses commonly used in phylogenetics. At BioHackathon 2009 [28], developers of the G-language project implemented web service interfaces so that the G-language functions would be available to workflows available in the popular workflow workbench application Taverna.

4.2.3.2 Proof-of-Concept

Some tools only demonstrated a concept's feasibility, but motivated the authors to develop a more sophisticated version after the hackathon. One example from the O|B|F CodeFest 2013 report was a visualization tool that made it possible to visualize an RNA sequence analysis while browsing the genome. This prototype later inspired a version of the tool that scheduled animation updates more efficiently, leading to smoother animations and more accurate windows.

Other useful community resources included:

4.2.3.3 Mailing Lists

In some instances, hackathon participants created mailing lists to sustain the energy of the hackathon after the event. For instance, after the NESCent hackathon for comparative methods in R, a hackathon that aimed to ensure compatibility and data flow between R packages, the participants created a mailing list for users and developers of the packages. Five years after the hackathon, the mailing list has 962 subscribers and an average of over 50 posts per month [11].

4.2.3.4 Documentation

Documentation is an additional important outcome of a hackathon. We observed that there is both documentation in the form of "records of the event" and documentation in terms of how to use the software that is produced. Examples of activities that participants document include use cases, the names of sub-groups, their progress in addressing the use cases, and future work [14].

Participants also created extensive documentation of the tools themselves, both for tools already in wide use, and for tools created at the hackathon. For instance, due to increased interest in using CloudBioLinux, a project providing machine images for bioinformatics on cloud computing platforms participants from the "Infrastructure management" group at Codefest 2013 created extensive documentation on the ReadTheDocs website [37].

4.2.3.5 Training and Tutorials on New Tools

Some developers we spoke with attended "bootcamps," short tutorials designed to help developers new to a toolkit to get acquainted with its basic design and coding principles (P7, P23, P36). We found evidence that these tutorials enabled some effective cross-project interactions. For instance, a developer from the HyPhy project added an interface to the Biopython codebase. In another example, a creator of PhyloXML contributed a NEXUS parser to the BioRuby project [32].

5. DISCUSSION

Below, we draw on our results to suggest how four different themes that cross-cut our work may have implications for organizing community code engagements: *task interdependence*, *ties*, *transparency of contributions*, and *appropriate mix of experts and novices*. We place these themes in the context of community growth and code contributions. We also discuss other possible forms of community code engagements, additional outcomes of interest beyond contributions and community growth, how the type of community results in different outcomes, and implications for Information and Communication Technologies (ICTs).

5.1 Community Growth

5.1.1 Task Interdependence

In the hackathon format, large groups of participants engage in face-to-face interactions. Face-to-face is an effective medium for highly interdependent tasks. Previous research by others has found that groups whose members work cooperatively on interdependent tasks tend to be more cohesive and committed to the group [18, 50]. Commitment may increase in these interdependent tasks as individuals see evidence that the group depends on them and values their work ([30], p. 85). Our findings indicate that a GSoC project, in contrast, involves a single student who works remotely on an isolated task. We speculate that independent tasks may make it difficult for students to understand the value of their contributions, which may lead to lower levels of commitment. It may also partly explain our finding that some students were unaware of how their code was being used after GSoC. Future study would investigate the relationship between task interdependence and community growth.

5.1.2 Ties

The benefits of completing highly interdependent work, however, may need to be balanced against the creation of strong ties in GSoC. Although GSoC tasks are isolated, our findings indicate that the longer, intensive mentoring facilitates the creation of strong ties between student and mentor. According to previous research, people who develop connections to others in a group work harder, do more, and tend to stick with the group longer ([30], p. 77). Therefore, the relative benefit of community growth will require more research to assess.

5.1.3 Transparency of Contributions

We suggest that the extent to which participants make their contributions visible to others will have a positive impact on community growth. We found that some Biopython GSoC students, for example, created blogs to promote their projects and posted links to their source-code on the blogs. Students also posted updates on their projects to the mailing list. These behaviors prompted other community members to comment on students' projects and help solve problems [47]. In contrast, the majority of Bioconductor GSoC students neither shared updates on their projects over the mailing list nor created materials promoting their projects. We found that Bioconductor community members (other than the project mentors) were often unaware of students' GSoC projects altogether.

Based on this evidence, we propose that if other community members do not see students' work, they will be less likely to provide feedback or offer suggestions for improvement. If this is true, students will not know if others value their work and may not feel strong enough commitment to stick with the community.

5.1.4 Appropriate Mix of Experts and Novices

Our findings indicate that community code engagements often provide opportunities for mentoring and learning, as novices and experts collaborate. GSoC mentors teach students about the codebase and community norms, and expose them to other community members through blog and mailing list posts. During hackathon tutorials, experts teach new contributors about a tool's codebase. According to Lave and Wenger [33], people join communities by being present and participating along with experts and learning while doing actual work, as "Legitimate peripheral participants" (LPP). Not only is it typical for peripheral participants to become core members through situated learning, it is apparently an important motivation for the learner to continue participating in the community [17].

There is presumably some ratio where mentoring and learning are most efficient, as the ratio influences the number of opportunities for situated learning. Future work around this topic is needed.

5.2 Code Contributions

5.2.1 Appropriate Mix of Experts and Novices

Before newcomers can contribute to open-source software, a socialization process is triggered [15, 17]. GSoC students, for example, go through a process of introducing themselves to the community, formulating project ideas, and learning the technical aspects of the code base with their mentors. As hackathon tutorials illustrate, even experienced developers must learn about how to contribute to other tools.

Seasoned *core members* of the community are likely to be the most expert contributors [12]. We found that these members are also aware of what contributions are needed. In GSoC, for instance, mentors often seed project idea lists. Moreover, student projects are heavily influenced by the vision of mentors. We speculate that, all other things (including number of participants and engagement duration) being equal, an engagement involving only core members would likely contribute more code than an engagement with more novices. A mixture of attendees including novices not only decreases the mean productivity of participants, but may cause the experts to devote time to assisting novices instead of coding.

This suggests a tension between the goals of *code contribution* and *community growth*: the greater proportion of experts present, the more code that will be produced, but the greater the ratio of novices present, up to some optimum, the more newcomers will join.

We suggest that **there are important tradeoffs involving both *appropriate mix of experts and novices* and *task interdependence***. An engagement in which many novice participants are included (up to some optimal number), and/or in which highly interdependent tasks are chosen, will contribute more to community growth but less to the codebase; and conversely an event in which experts work on independent tasks will be likely to grow the source code without doing as much to grow the community.

5.3 Hybrid Forms of Community Code Engagements

There are likely hybrid forms of engagements worth exploring that mix aspects of GSoC and hackathons. For instance, one issue we raised in this work is that GSoC students are seldom exposed to other students, mentors, and the larger community. A possible variation on GSoC would be, at the midpoint of the project, to send the student to a community conference. In addition to receiving feedback on their projects, the student could get exposure to the networking and relationship building benefits of hackathons, such as hearing about job opportunities and meeting potential users of their software. On completion of the GSoC project, they may feel more connected to the community, feel that others value their work, and perhaps be more likely to stick around.

As another example, a variation on the hackathon format would be to invite students to a hackathon and pair them off with more experienced members of the community. This configuration might be a way for the engagement designer to strike a balance between code contributions and community growth. As they work side-by-side with mentors in the sub-group, students would not only learn by doing, but also get a sense of real issues that matter to the

community that they aspire to join. Working on interdependent tasks would enhance students' perceptions that their work has value. Mentors could delegate simpler tasks to students therefore freeing them up to work on more difficult tasks. Students would see how their contributions matter in the "big picture" while mentors would be able to devote more of their time to coding.

5.4 Other Outcomes

We suggest that there are two important outcomes in addition to community growth and code contributions: visibility of community needs, and training.

5.4.1 Visibility of Community Needs

Both GSoC and hackathons provide the community with an occasion to identify, discuss, and prioritize needs in a way that is generally visible to everyone. The creation of the idea list in GSoC facilitates discussion with potentially new community members and existing community members, who may normally not have the interest or need to engage one another. The agenda setting phase of the hackathon facilitates real-time interactions and discussions with community members who may normally have a willingness to collaborate, but who otherwise face obstacles of geographical and temporal dispersion. Participants can therefore establish a common vocabulary for talking about the work and develop shared goals before development begins. These mechanisms may play an important role in bringing the community together around common goals, regardless of what is accomplished by any particular engagement.

5.4.2 Training

We find that for some students, GSoC is not only their first exposure to the project's codebase, but also to software engineering practices in general such as versioning, unit testing, and object-oriented programming. During hackathons, participants receive training on other software tools and projects of interest. This training seems an important component for the sustainability of scientific software, since new generations of newcomers will need a certain set of technical skills to fill the roles of the original authors. Unfortunately, research shows that scientists tend to undervalue important software engineering concepts like modularity, test-driven development, versioning, and tend to underestimate the amount of time required to develop the software [40]. This not surprising, as scientists are trained in their domain of science, not software engineering. Future work should therefore examine how to structure engagements around optimizing for training, not just code contributions and community growth.

5.5 Impact of Community Type on Outcomes

Our findings suggest that situated learning, a concept from communities of practice [33], may help explain GSoC outcomes. Students learn throughout the process, from introducing themselves to the community, proposing project ideas, discussing project plans, and resolving issues related to the code they write, all with support from their mentors and other community members. In general, upon project completion, their code is added to the codebase. Afterward, they may continue to develop new features (e.g., P4, P5, P8), mentor future students (e.g., P4, P5), or both (e.g., P4, P5). Students thus *become* contributors, they do not simply *learn about* how to contribute.

The hackathons in our sample, in contrast, had more of a flavor of scientific software communities of interest. GSoC involved pairing up newcomers who had scientific domain knowledge and at least some knowledge of software development with mentors, whereas hackathons involved experienced developers working with end user scientists who had the domain knowledge. The presence of both groups was mutually beneficial; end users played an important role in determining requirements for the software (i.e., providing use cases), and the developers played an important role in demonstrating what software was possible using prototypes and proof-of-concepts. Developers often ran tutorial sessions to teach other developers, which are examples of knowledge being codified and then transferred to others, not situated learning. As we discussed previously, there may be promise, however, for facilitating situated learning by investigating hybrid forms of community code engagements.

5.6 Implications for Information and Communication Technologies (ICTs)

Although community code engagements have several positive outcomes, we also found evidence of many technological challenges that participants faced. Among them, there are two major issues that we discuss here.

Ranking Proposals. During the student application period in GSoC, mentoring organizations receive a huge number of project proposals from students. Since only a few can be accepted, the task of ranking these proposals becomes important. Also each proposal requires a mentor to be assigned, however, the availability of mentors is usually limited. Therefore, the use of software tools can assist in the process of ranking proposals, sharing with other members and assigning mentors. The Biopython community does this by having lots of discussions on the mailing list. One Biopython participant (P1) mentioned that this results in a flurry of emails and is often difficult to keep up. The participant suggested that developing a tool with a Reddit[2] like interface where members can up / down vote proposals, sort, comment and share them, could facilitate this process.

Video Chats as a Substitute for Face to Face Interaction. We found that in GSoC, students and mentors almost always coordinated remotely. Some of them used emails whereas others relied heavily on real time video chat tools such as Google Hangouts and Skype. While some mentors (P29) found it comfortable to communicate via emails when students were able to work independently, other students (P48) and mentors (P26) felt they needed more face-to-face interaction. In the latter cases, the use of video chat technologies was found to be an appropriate substitute for working from the same physical location. One participant (P48) acknowledged that this helped them get a better sense of what they were trying to communicate and therefore, sped up the process.

6. CONCLUSION

In this work we examined two community code engagements: Google Summer of Code and hackathons. We sought to understand the range of outcomes these engagements produce and the underlying practices that lead to those outcomes. We found that in GSoC, the vision and experience of core team members influences project selection and the mentoring process facilitates creation of strong ties. Most GSoC projects result in stable features. The agenda setting phase of hackathons reveals high priority issues perceived by the community, and social events create weak ties. Most hackathons result in promising prototypes rather than finished tools. Our findings point to several themes and tradeoffs around community code engagement design that we hope to explore in future empirical work.

[2] http://www.reddit.com/

As is common with case studies, the generalizability of our results is limited. On the one hand, some elements from community code engagements seem applicable to other types of collaborative groups, not just open-source software. For instance, mentorship, which facilitates the socialization of newcomers, seems useful to explore in Wikipedia, where contributions from newcomers are disproportionally rejected due to not following standard policies [23]. Moreover, articulating and prioritizing user needs seems fundamental to eliciting contributions in any community, because contributors will know what to do. On the other hand, these elements are likely impractical for software shared only within local laboratories, tailored to a particular purpose, and limited to a few developers and users. Numerical simulations, for example, are difficult to make generally useful, and many scientists are reluctant to share, or open up development of the code, lest others use it incorrectly and produce spurious results [45]. In addition, different scientific fields may value individual skill and reputation in developing software over collective achievements. Future work could thus elaborate on the conditions under which community code engagements are appropriate and likely to have impact.

7. ACKNOWLEDGMENTS

This work was supported by a grant from the Alfred P. Sloan Foundation, the Google Open Source Programs Office, and NSF awards IIS-1111750, SMA-1064209, and ACI-0943168. We thank our participants for taking time out of their busy schedules to collaborate with us.

8. REFERENCES

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. 1990. Basic local alignment search tool. *Journal of Molecular Biology*. 215, (1990), 403–410.

[2] Atkins, D., Droegemeier, K., Feldman, S., Garcia-Molina, H., Klein, M., Messerschmitt, D., Messina, P., Ostriker, J. and Wright, M. 2003. *Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure*.

[3] Attracting and Retaining Participants: 2010. *http://strategy.wikimedia.org/wiki/Attracting_and_retaining_participants*. Accessed: 2014-02-02.

[4] Biopython Google Summer of Code: *http://biopython.org/wiki/Google_Summer_of_Code*. Accessed: 2014-06-13.

[5] Blatecky, A. and Messerschmitt, D. 2005. *Planning for Cyberinfrastructure Software*.

[6] Christopherson, L., Idaszak, R. and Ahalt, S. 2013. *Developing Scientific Software through the Open Community Engagement Process*.

[7] Cock, P.J. a, Antao, T., Chang, J.T., Chapman, B. a, Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B. and de Hoon, M.J.L. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics (Oxford, England)*. 25, 11 (Jun. 2009), 1422–3.

[8] Codefest: *http://www.open-bio.org/wiki/Codefest*. Accessed: 2014-06-13.

[9] Colwell, R. 2000. Information Technology: Ariadne's Thread Through the Research and Education Labyrinth. *Educause*. June (2000), 15–18.

[10] Corbin, J. and Strauss, J. 2008. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage.

[11] Cranston, K. and Evolutionary, N. 2013. A grassroots approach to software sustainability. *Proc. Workshop in Sustainable Software for Science: Practice and Experience (WSSSPE)* (2013).

[12] Crowston, K., Wei, K., Howison, J. and Wiggins, A. 2012. Free/Libre open-source software development: what we know and what we do not know. *ACM Computing Surveys*. 44, 2 (Feb. 2012), 1–35.

[13] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (2012), 1277–1286.

[14] Database Interop Hackathon: *https://www.nescent.org/wg/evoinfo/index.php?title=Database_Interop_Hackathon*. Accessed: 2014-06-13.

[15] Ducheneaut, N. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*. 14, 4 (Jul. 2005), 323–368.

[16] Fabri, A. and Pion, S. 2009. CGAL: the Computational Geometry Algorithms Library. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2009), 538–539.

[17] Fang, Y. and Neufeld, D. 2009. Understanding Sustained Participation in Open Source Software Projects. *Journal of Management Information Systems*. 25, 4 (Apr. 2009), 9–50.

[18] Gaertner, S. and Dovidio, J. 2000. Reducing intergroup conflict: From superordinate goals to decategorization, recategorization, and mutual differentiation. *Group Dynamics: Theory, Research, and Practice*. 4, 1 (2000), 98–114.

[19] Gentleman, R.C. et al. 2004. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*. 5, 10 (Jan. 2004), R80.

[20] Google Summer of Code: 2013. *https://developers.google.com/open-source/soc/?csw=1*. Accessed: 2014-01-31.

[21] Granovetter, M. 1973. The Strength of Weak Ties. *American journal of sociology*. 78, 6 (1973).

[22] Hackathon: *http://en.wikipedia.org/wiki/Hackathon*. Accessed: 2014-02-12.

[23] Halfaker, A., Kittur, A. and Riedl, J. 2011. Don't Bite the Newbies: How Reverts Affect the Quantity and Quality of Wikipedia Work. *Proceedings of the 7th International Symposium on Wikis and Open Collaboration* (2011), 163–172.

[24] Henri, F. and Pudelko, B. 2003. Understanding and analysing activity and learning in virtual communities. *Journal of Computer Assisted Learning*. October 2002 (2003), 474–487.

[25] Howison, J. and Herbsleb, J.D. 2013. Incentives and integration in scientific software production. *Proceedings of the ACM 2013 conference on Computer Supported Cooperative Work* (New York, New York, USA, 2013), 459–468.

[26] Howison, J. and Herbsleb, J.D. 2011. Scientific software production : incentives and collaboration. *Proceedings of the ACM 2011 conference on Computer Supported Cooperative Work* (2011), 513–522.

[27] Jirotka, M., Procter, R., Rodden, T. and Bowker, G.C. 2006. Special Issue: Collaboration in e-Research. *Computer Supported Cooperative Work (CSCW)*. 15, (Sep. 2006), 251–255.

[28] Katayama, T. et al. 2011. The 2nd DBCLS BioHackathon: interoperable bioinformatics Web services for integrated applications. *Journal of biomedical semantics*. 2, 4 (Jan. 2011), 1–18.

[29] Katayama, T. et al. 2010. The DBCLS BioHackathon: standardization and interoperability for bioinformatics web services and workflows. *Journal of Biomedical Semantics*. 1, 8 (2010), 1–19.

[30] Kraut, R.E. and Resnick, P. 2011. *Building Successful Online Communities*. MIT Press.

[31] Von Krogh, G., Spaeth, S. and Lakhani, K.R. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*. 32, 7 (Jul. 2003), 1217–1241.

[32] Lapp, H., Bala, S., Balhoff, J.P., Bouck, A., Goto, N., Holland, R., Holloway, A., Katayama, T., Lewis, P.O., Mackey, A.J., Osborne, B.I., Piel, W.H. and Pond, S.L.K. 2007. The 2006 NESCent Phyloinformatics Hackathon: A Field Report. *Evolutionary Bioinformatics*. 3, (2007), 287–296.

[33] Lave, J. and Wenger, E. 1990. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.

[34] OpenScience Software: *http://openscience.org/links/*. Accessed: 2014-01-31.

[35] Phylotastic1: *http://www.evoio.org/wiki/Phylotastic1*. Accessed: 2014-06-13.

[36] Pico, A.R., Kelder, T., van Iersel, M.P., Hanspers, K., Conklin, B.R. and Evelo, C. 2008. WikiPathways: Pathway Editing for the People. *PLoS Biology*. 6, 7 (2008), e184.

[37] Proteomics Data Analysis with CloudBioLinux: *http://mass-spec-data-analysis-with-cloudbiolinux.readthedocs.org/en/latest/*. Accessed: 2014-06-13.

[38] Ribes, D. and Lee, C.P. 2010. Sociotechnical Studies of Cyberinfrastructure and e-Research: Current Themes and Future Trajectories. *Computer Supported Cooperative Work (CSCW)*. 19, 3-4 (Sep. 2010), 231–244.

[39] Schroeder, W.J., Martin, K.M. and Lorensen, W.E. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. *Proceedings of the 7th conference on Visualization* 93–100.

[40] Segal, J. 2009. 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, 5-6 (Sep. 2009), 581–606.

[41] Shah, S. 2006. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*. 52, 7 (2006), 1000–1014.

[42] Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B. and Ideker, T. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*. 13, 11 (Nov. 2003), 2498–504.

[43] SocioCultural Research Consultants, L. 2013. Dedoose Version 4.5, web application for managing, analyzing, and presenting qualitative and mixed method research data.

[44] Stewart, C.A., Almes, G.T., McCaulay, S. and Wheeler, B.C. eds. 2010. *Cyberinfrastructure Software Sustainability and Reusability*. Indiana University.

[45] Sundberg, M. 2010. Organizing Simulation Code Collectives. *Science studies: an interdisciplinary journal for science and technology studies*. 23, 1 (2010), 37–57.

[46] SWAT4LS 2012: *http://www.w3.org/wiki/HCLS/SWAT4LS2012/Hackathon*. Accessed: 2014-06-13.

[47] Trainer, E., Chaihirunkarn, C. and Herbsleb, J. 2013. The big effects of short-term efforts: A catalyst for community engagement in scientific software. *Proc. Workshop in Sustainable Software for Science: Practice and Experience (WSSSPE)*. (2013).

[48] Trainer, E.H., Chaihirunkarn, C., Kalyanasundaram, A. and Herbsleb, J.D. 2015. From Personal Tool to Community Resource: What's the Extra Work and Who Will Do It? *Proceedings of the ACM 2015 Conference on Computer Supported Cooperative Work & Social Computing* (2015), to appear.

[49] Use Cases: *http://informatics.nescent.org/wiki/UseCases*. Accessed: 2014-06-13.

[50] Worchel, S., Rothgerber, H., Day, E., Hart, D. and Butemeyer, J. 1998. Social identity and individual productivity within groups. *British Journal of Social Psychology*. 37, (1998), 389–414.

[51] Working towards Sustainable Software for Science: Practice and Experiences: 2013. *http://wssspe.researchcomputing.org.uk*. Accessed: 2014-01-31.