Share: A Programming Environment for Loosely Bound Cooperation

Yannick Assogba
MIT Media Lab
20 Ames Street, Cambridge MA, USA
yannick@media.mit.edu

Judith Donath
Harvard University Berkman Center
23 Everett Street, Cambridge, MA, USA
jdonath@cyber.law.harvard.edu

ABSTRACT
We introduce a programming environment entitled *Share* that is designed to encourage loosely bound cooperation between individuals within communities of practice through the sharing of code. Loosely bound cooperation refers to the opportunity community members have to assist and share resources with one another while maintaining their autonomy and independent practice. We contrast this model with forms of collaboration that enable large numbers of distributed individuals to collaborate on large scale works where they are guided by a shared vision of what they are collectively trying to achieve. We hypothesize that providing fine-grained, publicly visible attribution of code sharing activity within a community can provide socially motivated encouragement for code sharing. We present an overview of the design of our tool and the objectives that guided its design and a discussion of a small-scale deployment of our prototype among members of a particular community of practice.

Author Keywords
Community, Cooperation, Collaboration ,Programming, Open Source, Visualization, Social Software, Computer Supported Cooperative Work

ACM Classification Keywords
H.5.3. Group and Organization Interfaces

General Terms
Human Factors

INTRODUCTION
Communities mediated by networked technology present opportunities for greater interplay between the individual and socio-contextual aspects of creative endeavor. Systems (both their social and technical components) such as Wikipedia, the development model of open source software such as GNU/Linux, and social computing experiments such as NASA Clickworkers [26] or the ESP game [13] are but a few examples of how the work of individuals collaborating at immense scale is synthesized to produce something greater than the sum of its parts. However for the most part these projects and many other ones like them are ones in which participants have a shared idea of what they are trying to achieve. Be it an encyclopedia or an operating system there is a shared goal that all participants are working towards. We seek to look at the design of systems that allow individuals, in this case programmers, to pursue independent goals yet still be able to help each other along the way. We refer to this form of collaboration as *loosely bound cooperation*.

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

BACKGROUND
Booch and Brown define a collaborative development environment (CDE) as a "virtual space wherein all the stakeholders of a project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task" [16]. This definition highlights the focus present in the collaborative development literature on collaboration organized along the axis of a single project or shared task. We see similar opportunities for collaboration, oriented not around shared goals, but rather shared resources within online communities.

In the social computing literature, Benkler [15] surveys various examples of distributed collaboration, two core examples he discusses are Wikipedia and the development of GNU/Linux. As in the description of collaboration given above, participants in these systems are bound together by a shared vision of what they are trying to achieve. Though they may have different reasons for participating, the goal of what they are working on, while co-created, is singular. To contrast, an example of a system that is constructed from more individualistic goals is *delicious.com* [3]. Rather than store web bookmarks locally on your computer, delicious.com allows you to store your bookmarks on their servers thus allowing you to access them from any computer with internet access. This goal is an individually oriented one — a user wants better access to their bookmarks. However in the context of a network of users, delicious.com is able to leverage this self-motivated behavior to provide added value for all users of the service. By allowing users to tag their bookmarks and by making them publicly searchable, delicious.com effectively provides a human filter on the larger internet. The public

nature of the bookmarks allows one to discover other users who are interested in similar web resources and see what else they are looking at. Wash & Rader [33] describe this extra value as a side effect of the individually motivated action to save bookmarks to a central server. In this system, users are not explicitly collaborating with each other, they are pursuing their own goals, however the service creates a form of cooperation between users.

The challenge we observe in the sphere of software development is that of how to support collaboration in the context of these new forms of radically distributed, often do-it-yourself, web-based communities. What does a CDE look like for individuals working on distinct projects? What are the factors that would encourage developers working on distinct projects to cooperate? If they are no longer trying to coordinate activity around a single project, what benefits may they gain from collaborating in the first place?

Loosely Bound Cooperation

We believe that collaboration within these web-based communities can be supported within a framework of what we term *loosely bound cooperation*. We hope to encapsulate in this framework the much weaker ties between members of the community (as compared to members of a typical software development team), as well the diversity of goals present among members of the community. We define loosely bound cooperation as *"a form of collaboration, often indirect, among members of a community that leaves them free to pursue their distinct individual goals, yet enables them to help each other along the way"*. Characteristics of loosely bound cooperation include:

- Participants do not have particular obligations (social nor contractual) to each other.
- Participants have distinct goals from each other (in the software example they are working on separate projects). Loosely bound cooperation does not disrupt their ability to pursue these goals
- Cooperation is often indirect and emerges primarily from artifacts produced out of the goals of the participants. We do not exclude more direct forms of assistance but rather emphasize a continuum of cooperative behavior that range from the very disconnected to the more engaged.
- Participants are members of a community of practice. Their shared practice provides the context for cooperation to take place and suggests artifacts that may be commonly useful to cooperators.

We already see examples of loosely bound cooperation between software developers when we consider snippet sharing sites such as gist.github.com [5] or DZone Snippets [4]. These sites allow programmers to post code snippets online and share them with others. They allow individuals to simply put useful bits of code 'out there' to be found by others who may in turn find them useful. There is very little binding 'collaborators' using DZone. Programmers also cooperate with each other on forums and question & answer sites such as StackOverflow [11]. Although these individuals do not share projects, by answering each others questions they do cooperate with each other. StackOverflow has built an elaborate reputation and reward system of points and badges that keep users engaged in the community. This problem of motivation is one that we think is particularly important in the context of loosely bound cooperation. What makes individuals with little in common willing to help each other? How can we design systems that reward this kind of behavior?

We believe that visualizations of community activity targeted at the community in question can play a role in motivating participation. Hill et al [25] introduce the notion of visible computational wear that allows digital artifacts to reveal their interaction and suggest that this 'wear' can help "mediate coordination and cooperation" by showing co-workers information on their use of the artifact. Erickson and Kellogg [18] extend this work to larger social systems with their notion of socially translucent systems that support visibility, awareness and accountability. Gilbert and Karahalios [21] also suggest that visualization can play a role in rewarding community production in open source software projects.

DESIGN PROPOSAL

We propose a novel programming environment geared towards supporting loosely bound cooperation between programmers within communities of practice [34]; our prototype is initially targeted at programmers using the *Processing* programming language [32], a language geared towards multimedia artists, designers and others interested in using code as a central part of their creative practice.

We chose Processing because we feel that the domain of computational art and design would be ideal for our exploration, as artists have strong individual goals, yet share a common toolset. The Processing community also has a rich history of cooperation and open working practice. and was designed early on to take advantage of web communities [32].

The primary method of cooperation that our tool supports is the sharing of code. Our tool shares all the code written in it with all members of the community as well as tracks its reuse, providing fine-grained attribution of where code came from as well as publicly visualizing the network of links created from the patterns of re-appropriation. In summary the system provides:

- Automatic Code Sharing. As code is written it is automatically distributed to all other users of the system.
- Tracking Copy & Paste. As code is re-appropriated its movement is tracked making it possible to see where any of the content in a particular file came from.

- Visualizing Relationships. The environment provides an interactive visualization of the entities within the system (users and code artifacts) and the relationships between them.
- Explicit reference and linking of artifacts. The system provides a means of making explicit references to other users or code artifacts for relationships that are not captured by the automatic copy-paste tracking (e.g. those indicating inspiration where no code actually moves).

Our hypothesis is that we can leverage public display of attribution to provide reward for, and motivate to participation in, code-sharing based cooperation between individuals who are in pursuit of independent goals.

RELATED WORK

Jazz [17] is an example CDE that provides support for ad-hoc teams organized around particular projects. It provides features such as chat, screen sharing, and status indicators to aid communication between software developers as well as awareness of team member activities and changes to source code. Jazz and other CDE's such as Microsoft Visual Studio Team System [8], Netbeans (with the optional collaboration module) [9] as well as studies on distributed software development such as that by Gutwin, Penner and Schneider [22] all focus on software 'teams' and 'groups' whose members have a strong need to maintain close awareness of what each other is doing. In the space we are exploring there are no teams and the usefulness of features such as traditional instant messaging can be called into question as the participants do not have direct dependent relationships and are under no obligation to help each other.

Scratch [29] is a programming language and community geared towards children that provides encouragement to share one's work. Youth are able to upload and download projects to and from the central Scratch website. When youth create projects based on those of others, the website automatically marks these projects as remixes, thus providing attribution for the original author. Monroy-Hernández [30] describes the ways in which the design of the Scratch website encourages participation in communal exchange. The Scratch website encourages uploading work to the site by highlighting works across various popularity metrics. Placing these projects prominently on the home page provides great reward for projects that successfully engage with other members of the community. However Scratch only allows for a project to have one 'ancestor', there is no easy way for an individual to incorporate code from multiple projects and if they are able to do so the system does not recognize the multiple contributions to a project.

GitHub [6] is a commercial code hosting service built upon the open source distributed version control system Git. It adds a social component to the code hosting facility provided by its competitors and provides the ability to track the alternate versions of a project that the sites users may create. However, the result of incorporating a number of components from distinct projects into a new work generally does not create a new 'version' of an existing project, and the process provided by GitHub does not support this aspect of code sharing practice that may be going on within a community of programmers. Similar to our comparison with Scratch, we seek to examine a design that enables code reuse at a finer level of granularity than that of an entire project.

OpenProcessing.org [10], is a community website for users of the Processing language to upload their works (along with source code) and put them on display for other visitors of the site. However unlike our work no explicit link is maintained to those who borrow code; unless the downloader leaves a comment, an uploader does not know if their code gets reused and likely cannot see how it was re-appropriated. Additionally we feel that OpenProcessing and other sites like it are set up more as exhibition spaces as opposed to workspaces; we will address this point further in our description of design goals below.

Our work is also informed by empirical research on the motivations of open source programmers. Among others, Ghosh [20], Lakhani & Wolf [27], Raymond [31], and Lerner & Tirole [28] have investigated individuals' motivations for participating in open source software. They have all identified socially oriented factors — i.e. factors that emerge from the context of working with others — that encourage participation. Importantly for us they all report that reputation or peer recognition for ones' contributions are strong motivating factors in encouraging participation. Share tries to capitalize on these factors to encourage code sharing among developers working on distinct projects.

SYSTEM DESCRIPTION

Design Goals

Our design goals revolved around the following considerations:

- Creating a good shared workspace vs. creating a good exhibition space. The environment and its mores should feel like a comfortable place for work in progress as opposed to being a place just for finished work.
- Non-disruptiveness. As much as possible we want to allow individuals, should they so desire, to work completely disengaged from the concept of working within a community — yet still be contributing to it. At the same time we want to provide a smooth continuum for increased engagement with the community. Thus interaction in Share is asynchronous and individuals are able to work online or offline.
- Non-competitiveness. Given our use of attribution as a reward mechanism, there exists a chance for the environment to become an overly competitive one, which we feel would be contrary to our goal of supporting cooperation and our desire to create a comfortable

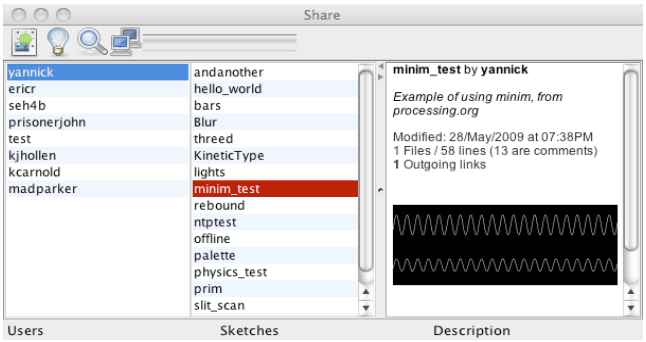

Figure 1. Share File Browser

workspace. We have mainly tried to focus on the more subtle (less competition oriented) displays of reputation.

System Architecture

Share is built using a client-server architecture with almost all the computation happening on the client. The server acts to provide authentication for clients and as a database through which documents and data files associated with projects are transferred. The client, a desktop application, is where the user does all of their programming.

Share Server

Share's server component consists of two main parts: a CouchDB database and a small ruby web application that controls authentication when pushing documents to the server. Apache CouchDB [2] is an HTTP accessible, schemaless, document-oriented database; CouchDB's document model fit well with the nature of our underlying data, and its web friendly architecture makes it amenable to easy integration with the client side components.

Share Client

File Browser

The file browser (fig. 1) is the first thing a user sees after logging in to Share and is their entry way to other parts of the software. The file browser allows users to look through other users' projects, and shows a description panel that displays metadata such as how big the project is (in files and lines of code), how many incoming links (files from which it has borrowed code from) and outgoing links (files to which it has contributed code) the project has, how many times it has been bookmarked as well as a screenshot from the application if the user has uploaded one. We also parse the comment at the top of the main file in a project to use as a description.

Editor

The editor is at the core of Share's functionality, it provides a means to edit code and also the mechanism to track the movement of code. Our code editor records attributes on each character of text such as which user wrote it, what document it originated from and in the case of code that was pasted in, the time and date that it was pasted. All documents and projects are given universally unique

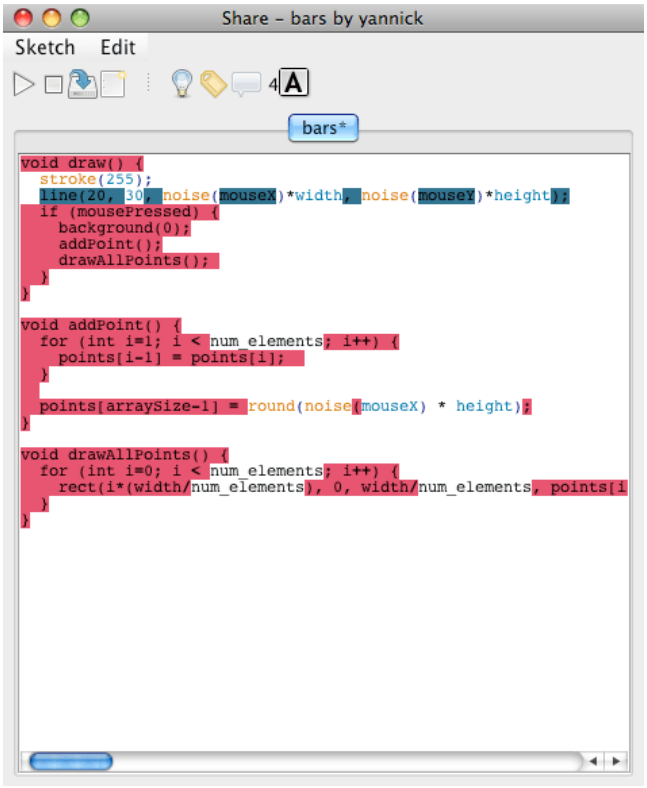

Figure 2. Share Code Editor in Source Highlighting Mode. Background color of text indicates which user it was borrowed from. Uncolored text is original to this document.

identifiers (UUIDs), the UUIDs are what the client uses to refer to the documents allowing them to be renamed freely without affecting our ability to track their content. The text is written to an XML based format that lets us persist and restore the attribution information. This representation allows very fine-grained representation (down to the character level) of where code came from. This allows the editor to perform code highlighting based on the *human* source of the code, as in fig. 2 where the background color of the text is determined by which user it came from (text with a clear background was created by the owner of this document). Upon startup colors are assigned to all users in the system and persist throughout a coding session, that color will consistently be used to represent that user, his projects and his code throughout the software. This source-highlighting mode can be toggled on and off. Another advantage of tracking reuse at this level of granularity is that it enables users to also see what was changed in a copied snippet. This could potentially be useful to beginners trying to understand which parts of code to tweak to make it work in a different context.

Explicit References

Share provides the ability to make explicit references to other projects. This is done using a special syntax directly in the file. This consists of using the *@saw* keyword and then giving a username/project pair to link to.

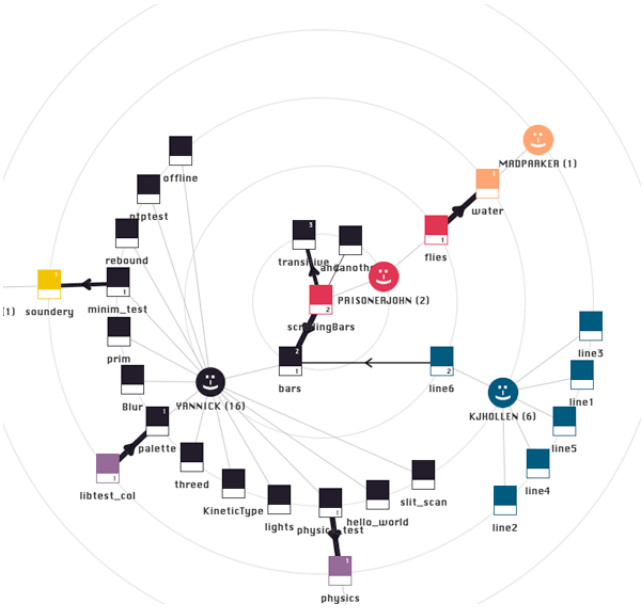

Figure 3. Share Network Browser in Radial Tree Mode. The circular icons represent users while the rectangular icons represent projects.

Search, Comments & Bookmarks

Share provides fulltext search of all the code in the repository through the use of the Lucene fulltext search engine [7] as well as a per-project comment thread accessible from the code editor. Share also allows individuals to bookmark projects that they find interesting. A more complete description of these and other subsystems can be found in [14].

The Network Browser

The network browser is an interactive visualization of the relationships between the projects and users Share. It acts as a form of visible (yet not explicitly ranked) reputation, as one can easily tell whether a project has contributed code to a lot of other projects. Given any project or user, a spanning tree is built of that entities' relationships in the overall network graph. The process of creating the tree from the more general graph potentially eliminates some of the links within the graph, however we feel that this representation more clearly shows the elements that are most closely related to the selected node. There are two visualizations provided by the network browser, the first is a radial tree view (fig. 3) where the selected entity is placed in the center, the algorithm used is a partial implementation of Yee et al's [35] layout algorithm for animated radial graphs that we ported from Jeffrey Heer's "prefuse" visualization library [24]. In this visualization successive rings display entities directly related to those on the inner ring. Arrowheads point in the direction that code traveled and the thickness of the arrow is proportional to the relative proportion of borrowed code in the borrowing project. We use color to relate project icons to the icons of their creator. The two are always rendered in the same color and this is

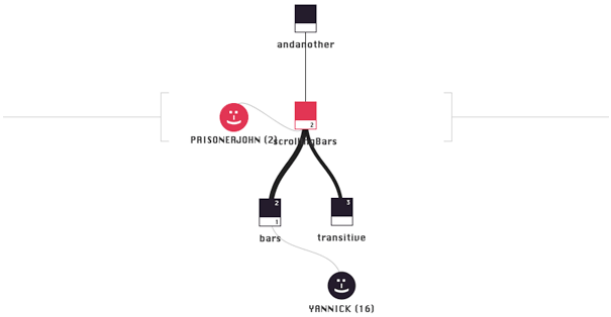

Figure 4. Share Network Browser in Simplified Mode

the same color used in the source highlighting view in the code editor. As the user clicks on nodes they are smoothly animated to the center and more distant nodes move closer to the center and additional nodes added, allowing the user to progressively move closer and closer to the leaves of the original tree (and also freeing us from trying to fit the entire graph onto the screen at once). This visualization is also aimed at supporting discovery of previously unknown resources. As you can see in fig. 3 there are a lot of nodes displayed that are only indirectly related to the selected (centered) node, while we did not implement any filtering to control the number of nodes shown in the visualization this would become necessary as more projects were created in the tool. One could use any number of metrics to filter out incidentally related nodes, including recency of edits, code similarity, popularity metrics and so on.

Users can toggle a second visualization that is much simpler than the radial tree view (fig. 4) which simply answers the question "what projects are contributing to and borrowing from this project". It thus shows elements that are only one step away from the selected node.

Synchronization

The synchronization subsystem is responsible for pushing local changes from the client and pulling new and updated documents from the server, it runs automatically every five minutes and only transfers new code (and project files) to and from the server.

Runtime

Share ships with the Processing compiler and runtime, however it is architected in such a manner that it can easily support other programming languages. When a user is running one of their own projects, our preprocessor for Processing code also adds a little bit of code to the project to enable the user to press a single key that will take a screenshot of their project and upload it to the server, this is used to generate the preview images shown in the file browser. For such a visually oriented community as the one we were targeting it was an important part of helping users explore each others work.

Security Concerns

One of the issues in a system like Share is that when a user runs another's sketch they are essentially running code from a stranger on the Internet. All programs run from Share are run in a sandbox managed by the Java Virtual Machine (JVM), the security policy we set when launching projects restricts them to a much safer set of operations.

Limitations

The chief limitations of the prototype as currently implemented revolve around scalability of the visualization and possibly the ability of users to find useful source code as the repository of projects grows in size. Future work would seek to investigate how these parts of the application could scale, however we are confident that solutions can be found. Programmers successfully find code on the Internet everyday, indeed it was this experience of 'programming-by-google' that partially inspired the authors to create this project. In addition to the full text search of the entire repository that we currently provide, research in code recommender systems such Codebroker [19] suggest that recommender technology could be used to both provide better search results and in pruning nodes in the visualization to favor displaying more semantically related projects. With regards to the visualization, there is plenty of visualization research into the display of large network graphs (e.g. Vizster [23]).

DEPLOYMENT AND USER FEEDBACK.

In order to evaluate our design we hosted a themed design/programming competition that was entirely electronically mediated. The purpose of the competition structure and theme was to scaffold the creation a small scale community of practice that would provide the loose associations and shared interests we would expect to see in larger communities of practice but do so in a manner that makes a shorter timeframe analysis practical. Participants in our competition were asked to create works 'Inspired by Pong', this was the only constraint given with respect to creative work. They were allowed a two week period over which to work on their submissions. We felt that the two week period would be sufficiently long to make apparent the asynchronous nature of interaction we designed Share to support. Of the sixteen participants that participated in the competition, eleven submitted pieces for consideration by the judges (participants could create as many pieces as they wanted during the competition but could only select one to submit for judging).

While prizes (two iPod Touches and two Arduino kits) were offered as incentive for extended participation and while a competition was used to recruit and encourage participants to actively use our software, a proviso was made that would award smaller prizes ($25 Amazon gift cards) to participants whose code was used in winning submissions. This meant that a person borrowing ones' code simply increases ones' chances of winning something, and was very much in keeping with the cooperative spirit of Share.

At the end of the competition, participants were asked to fill out a questionnaire on various aspects of their experience; eleven of the sixteen participants completed the survey. The investigator also interacted with the participants throughout the course of the event. The results discussed in this section come from two main sources, the metadata on code-sharing collected by the software (data from all sixteen participants) and the participants' responses to the questionnaire (from the eleven respondents).

Recruitment and Participant Demographics

Individuals from the Processing community were recruited over the Internet and invited to volunteer for the study. This does imply some self-selection bias with regards to willingness to share code, however we do not feel that this is a problem as we explicitly situate our work within the sharing economy, that is to say we are not contrasting it with proprietary models but rather aim to support those already participating in sharing economies. The participants were physically distributed across different parts of the world, including Europe, Asia and the USA. An Internet Relay Chat (IRC) server was set up for the participants to use, however due to time zone differences, there were never that many people in the chat room at once. The chat room served as a source of live technical support, both for programming techniques and issues with the software itself. Participants ranged across all levels of experience from newcomers to programming to long term experts; and included hobbyists, students in art, design and architecture as well as professional designers and artists.

Usage data

65 projects were produced by the 16 participants over the two week period, 12 of these were removed from this analysis because they were duplicate projects created by their owners to overcome implementation bugs in the software (during the course of the competition some files became corrupted and were no longer editable by their owners, they were thus duplicated), thus 53 projects were used in this analysis, with each user creating an average of 3.31 projects (standard deviation=2.84, median=3). These projects include the main submissions the participants were working on as well as many small sketches to test a particular idea or piece of code. We include these 'side' sketches because we feel that they are an important part of the process of coding, and a valuable piece of what users get to see when looking at each others work. Our presentation of this data is mainly to indicate the level of activity in Share over the two-week period. Across all 53 projects the average percent of borrowed code in each project was 13.9%. With 32.1% of projects having at least one incoming link (borrowed code from another user's project) and 60.4% having at least one incoming or outgoing link (having borrowed from or contributed to another user's project). As shown in fig. 5, the distribution of borrowed code follows a power law distribution.

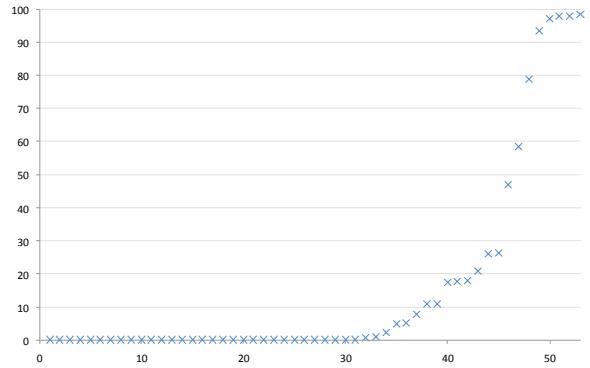

Figure 5. Percentage of code borrowed from other users across all 53 projects.

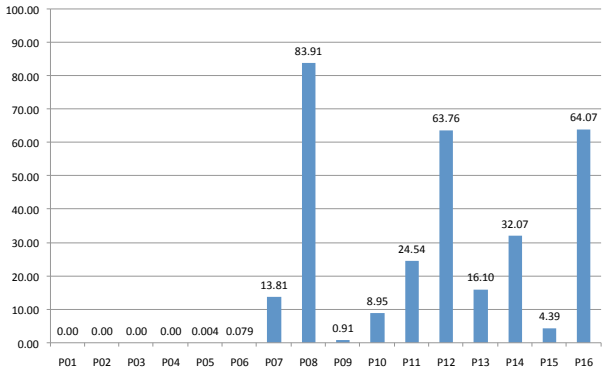

Figure 6. Percentage of borrowed code across all 16 participants.

Figure 6 shows the distribution of code borrowing activity across the 16 participants. The data suggests that a number of participants (P08, P12 and P16) made a deliberate strategy of 'remixing' others work, or at the very least found a very good base from which to continue their projects.

We can also look at the data from projects that have at least one incoming or outgoing link (the connected components of the network). These measures do overestimate the 'usefulness' of the code base (in terms of the amount code that a user was directly able to re-use). 32 of the 53 projects have at least one incoming or outgoing link and form 5 connected components in the overall graph. The average percentage of code borrowed from other users among these projects is 21.0%, the distribution is the same as the one in fig. 5 except that is has a shorter 'tail'.

This data indicates that there was reasonable usage of the features provided by Share and is in line with what we would expect; given that the projects are independent we would not actually expect to see large percentages of borrowed code in most cases.

In terms of what kind of code was reused, we observed the code for constructing the basic mechanics of a Pong game spread the most among projects; these include things like collision detection and physics simulations or the code used to control the 'paddles' common to pong games. Also a number of techniques particular to small sets of projects would originate from a particular user and spread to a few others, this included calculations for geometry and movement on circular paths (a number of project put a 'circular' spin on pong).

Survey Responses

Our hypothesis was that automatic tracking and attribution of code would lower barriers to sharing code and provide encouragement to share code with others. The data and quotes in this section come from the eleven responses to the survey that we received.

Our survey focused on a number of different aspects of participants' experience using Share. These include: whether Share reduces individuals barriers to openly sharing code, the value of the automatic attribution, whether individuals felt more productive or creative when using Share, the usefulness of the visualization in discovering code and understanding community activity, and whether the feature set of Share was disruptive to users' regular programming practice.

We asked users how they felt about the automatic attribution provided by Share (i.e. tracking reuse of their own code), all respondents responded positively to this feature, saying

"When releasing code, you don't need really know if it has its own life beyond your project. It's stimulating to see it travel around."

"I love the code tracking and highlighting so that I can follow the chain back to see how my implementation of something I copied can improve."

"It helps that it does the attribution for you, so you don't have to remember which snippet came from where, or be constantly documenting it, which can interrupt the flow of coding."

"It is nice to be able to trace ideas and code. The @saw tag was useful for me, because it allowed me to write notes for myself so that I would remember where I saw an idea and how they implemented it."

"It is a good way to learn about other people through what they do. It is also a good way to see how helpful/useful the stuff you produce is."

"It's a good thing and makes you feel like you are not working on your own but collaborating with many people without the sense of "being stealing" someone else's stuff"

This is exactly the response we were hoping for; there was value in simply seeing your contributions being used by others; individuals are encouraged to contribute code to the commons because they can see if it takes on a life beyond its original use. There is also pragmatic value in seeing what had been done with it as one can keep an eye out for improvements. The display of attribution also increases the sense of community and reduces anxiety around issues of 'stealing' other peoples' code. We also see that it supports

something people are doing already, i.e. documenting their sources when they borrow code.

We asked users directly whether they felt that the *attribution features of Share lowered their barriers to sharing code*, their responses to this question on a 5 point Likert scale are displayed in fig 7.

In addition to being able to see where their code went, users reported that Share reduced their barriers to sharing code because the continual and automatic uploading of their code reduced the anxiety burden of making their work visible to others. We had asked users about what prevented them from sharing code prior to using our tool and one common response was a feeling that their code was not 'good enough' to be shown to other people. Share takes this burden out of the users hands and a number of users were grateful for this.

With regard to using the visualization to track the movement of *other peoples' code*, 6 out of 11 respondents said they found it useful. The main use of the visualization with regard to other peoples' code was seeing what code was popular within the community and thus warranted further investigation, respondents also found watching the changes in the network visualization gratifying as a sign of the presence of other users thus increasing the sense of community among participants. Positive responses include:

"Well, when I saw that a lot of people were borrowing from a particular [project], I'd check out that person's code, because there must be something cool in there if that many people are using it."

"Yes, it made me concentrate on this traveling code. I was more interested by sketches that had connections over sketches that hadn't [...] it's something I liked, to see day after day, the network building itself."

"I liked the visualization as a way to gauge overall productivity and activity of the community but I think it was too abstract to tell me much about code."

The negative responses to this question were not very detailed, with respondents saying that they simply did not use the feature that much. Users suggested that the visualization could have been more helpful if it more quickly allowed for an individual to get more information about that project other than what it was connected to. The end of the last comment quoted above does point to an opportunity to encode more information about a project in the visualization itself, possibly through parameterizing the design of the icons representing projects with project related features. Something we did not see was the use of the network visualization to discover previously unknown resources, this is not too surprising due to the small number of participants. There was little that could not be discovered by browsing through the lists in the file browser. We suspect that the utility of the network visualization in this regard would increase as the size of the community using Share grows.

We asked participants whether they felt that the feature set in our tool made them more productive (able to do things more quickly or more creative (encouraged them to do things they otherwise may not have thought of), responses are shown in fig 8 and 9 respectively.

When asked to elaborate on how it resulted in increased productivity or creativity, we received the expected response that simply having a repository of code to draw from helped people get started more quickly, or otherwise more quickly solve their own problems. Participants also enjoyed seeing how others approached the same problem and found some inspiration for their own work. While we are unsure if this last point is more an effect of the competition's pong constraint, making it more likely to see something that gives you an idea; we are confident that it would be similarly useful in less constrained settings. Participant elaborations included:

"I'm a very beginner and Share let me have a look at other people's work and learn from them and their codes"

"Looking at what others are doing was a good starting point for generating ideas."

"We were some[times] to be in front of the same problems (dealing with collisions, or mouse control for example) I found other paths helpful to deal with these."

"I feel my abilities expanded when I could view everybody's code. I could see other people's solutions to problems arising in my own coding."

"It opened a vast space of ideas, of different approaches, that questioned mine. Seeing some others build their sketch day after day was very interesting too, changes they made, it was like seeing the living process of a creative idea."

However at least one respondent found the visibility of the other projects somewhat overwhelming, this individual was a bit intimidated by some of the work he saw being produced, saying,

"Though the wealth of code and projects is certainly inspiring, it's also a little overwhelming. Seeing everyone else's ideas made mine seem pale in comparison. Then again, I've been in a bit of a creative slump lately."

This last comment underscores the importance of creating a comfortable space for participants at all levels of experience, and is one of the reasons we avoid explicit ranking systems and leader-boards as we feel those type of reputation systems would alienate less experienced or less confident users.

Another user did mention the issue of signal vs. noise in browsing through the repository, saying *"the number of 'dead' sketches made it hard to fully discover the real diamonds"*. This is a common issue in systems providing access to 'user generated content' but solutions to this kind of problem (such as tagging systems) continue to be developed in numerous online spaces and is certainly

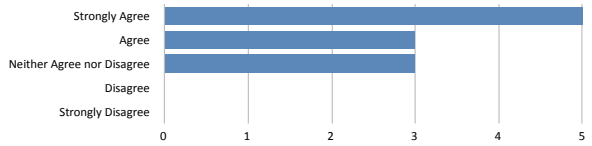

Figure 7. Responses to the question "The attribution methods provided by Share lower my barriers for sharing code"

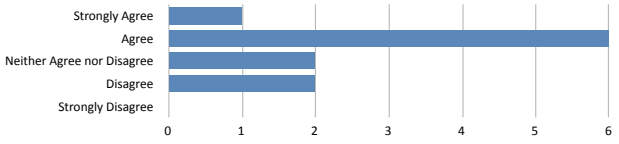

Figure 8. Responses to the question "The features provided by Share increased my ability (made it easier) to address the task"

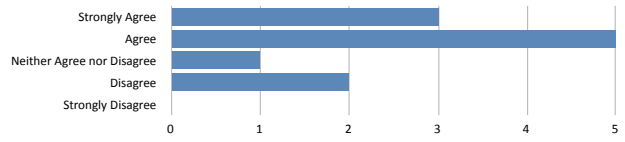

Figure 9. Responses to the question "The features provided by Share increased my creativity in addressing the task at hand."

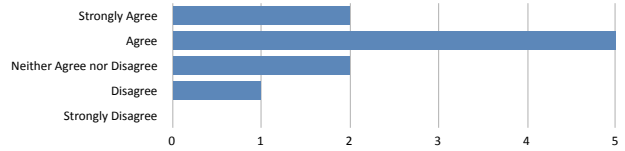

Figure 10. Participant responses to the question "When working in Share, I feel I am able to work independently and unencumbered by the notion of working within a community"

something that warrants further consideration in future iterations.

One of our goals in the design of Share was to minimize the disruption to regular work practice, when we asked participants whether they were able to work "unencumbered by the notion of working within a community", users generally felt that they were able to maintain their independence (fig. 10).

DISCUSSION

Our initial hypothesis is supported by the feedback we got from our users on how they felt about our automatic attribution as well as that indicated in fig 7. While our results share much in common with the literature around projects like Scratch, the environment we design for is different in a number of ways that provide alternate avenues for exploration. Firstly, in Scratch or on OpenProcessing.org the *work* is the primary thing that is being shared. That is to say, the goal of a user uploading a scratch project is not necessarily to *just share code* but primarily to share their creative output. We believe our design is more code centric; rather than an being exhibition space to display finished work, Share is an open workspace where unfinished code and ideas are open to all. In Share our continuous and automatic uploading of code took the burden of 'sharing', and thus selecting what is worthy, off the shoulders of the users and aided in making the system amenable to works in progress. We believe the difference between exhibition space and workspace is important to future design of collaborative tools for web based communities. Secondly the mechanisms available to provide reward differ slightly, while we do not have a front page through which we can leverage popularity to reward contribution, our user feedback suggests that we are still able to reveal enough of the social history of interaction around a particular users work to provide them value. We believe that the design and evaluation of less overtly competitive reputation systems is also an interesting area for future research and in longer term deployments of our system.

Our feature set also supports the existing programmer practice of documenting one's sources. In a similar manner to TrackBack [12] on blogging platforms or document repositories such as arXiv [1], our tool announces to content producers how and where their content is reused. As far as we know there is no existing trackback like system for programmers who share code and we believe that our tool makes an argument (and suggests some techniques) for creating a set of online services and programming conventions that would make it easier to create trackback for programmer communities, even without a specialized tool like Share in communities where switching costs would be prohibitive.

Supporting better documentation of sources also relieves some of the issues around the feeling that one is 'stealing' code; we feel that negotiating these issues around ownership and the relationship between the contributor and the borrower are important functions that tools for distributed web based cooperation can provide.

CONCLUSION

This paper has articulated the practice of loosely bound cooperation, in which individuals are able to pursue distinct, independent goals yet assist each other along the way and has described the design of a novel programming environment that facilitates this form of cooperation among members of a community of practice. The automatic tracking and public display of attribution provided by our tool contributes to positive feelings among the participants, as they feel recognized for their creative work and community contribution. Users also feel more at ease with reusing the work of others without feeling like they are stealing, and most of our users affirmed that it reduced their barriers to publicly sharing code. Individuals were also able to track downstream changes to contributions they had made and confirmed the pragmatic usefulness of doing so as well as the encouragement provided by seeing something they had created take on a life beyond their own projects. Share also alleviates some of the anxiety associated with

'publishing' one's work as it is constantly uploading works in progress for all users of the system.

REFERENCES

1. arXiv.org e-Print archive. http://arxiv.org/.
2. Apache CouchDB: The CouchDB Project. 2008. http://couchdb.apache.org/.
3. Delicious. http://delicious.com/.
4. DZone Snippets: Store, sort and share source code, with tag goodness. http://snippets.dzone.com/.
5. Gist - GitHub. http://gist.github.com/.
6. GitHub. http://github.com/.
7. Apache Lucene - Overview. 2006. http://lucene.apache.org/java/docs/.
8. Microsoft Team System. http://msdn.microsoft.com/en-us/teamsystem/default.aspx.
9. NetBeans. http://netbeans.org/.
10. OpenProcessing - Share Your Sketches! http://www.openprocessing.org/.
11. Stack Overflow. http://stackoverflow.com/.
12. TrackBack Technical Specification. http://www.sixapart.com/pronet/docs/trackback_spec.
13. von Ahn, L. and Dabbish, L. Labeling images with a computer game. *Proc. 2004 Conference on Human factors in computing systems*, ACM Press (2004), 326, 319.
14. Assogba, Y. Creative Networks: Socio-Technical Tools for Loosely Bound Cooperation. 2009. http://smg.media.mit.edu/papers/yannick/yannick_assogba_thesis_final.pdf.
15. Benkler, Y. The Wealth of Networks. Yale University Press, 2007.
16. Booch, G. and Brown, A.W. Collaborative development environments. *Advances in Computers* 59, (2003), 2–29.
17. Cheng, L.-T., Hupfer, S., et al., "Jazz: a collaborative application development environment," *ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, pp. 102-103, Anaheim, CA, USA, 2003.
18. Erickson, T. and Kellogg, W.A. Social translucence: an approach to designing systems that support social processes. *ACM Transactions on Computer-Human Interaction* (2000), 59–83.
19. Fischer, G. and Ye, Y. Personalizing Delivered Information in a Software Reuse Environment. In *Proc. 8th International Conference on User Modeling*, (2001), 178--187.
20. Ghosh, R.A. Understanding free software developers: Findings from the FLOSS study. *Perspectives on free and open source software*, (2005), 23–45.
21. Gilbert, E. and Karahalios, K. CodeSaw: A social visualization of distributed software development. *Lecture Notes in Computer Science* 4663, (2007), 303.
22. Gutwin, C., Penner, R., and Schneider, K. Group awareness in distributed software development. *Proc. 2004 ACM conference on Computer supported cooperative work*, (2004), 72–81.
23. Heer, J. and Boyd, D. Vizster: Visualizing online social networks. *Proc. 2005 IEEE Symposium on Information Visualization,* (2005), 33–40.
24. Heer, J., Card, S.K., and Landay, J.A. Prefuse: a toolkit for interactive information visualization. *Proc. SIGCHI conference on Human factors in computing systems*, (2005), 421–430.
25. Hill, W.C., Hollan, J.D., Wroblewski, D., and McCandless, T. Edit wear and read wear. *Proc. SIGCHI conference on Human factors in computing systems,* (1992), 3–9.
26. Kanefsky, B., Barlow, N.G., and Gulick, V.C. Can Distributed Volunteers Accomplish Massive Data Analysis Tasks. *Lunar and Planetary Science* XXXII, paper 1272, (2001).
27. Lakhani, K. and Wolf, R. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *Perspectives in Free and Open Source Software*, MIT Press, Cambridge, MA, (2005).
28. Lerner, J. and Tirole, J. The scope of open source licensing. *Journal of Law, Economics, and Organization* 21, 1 (2005), 20–56.
29. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. Scratch: a Sneak Preview. *Proc. Second International Conference on Creating, Connecting and Collaborating through Computing*, 2004.
30. Monroy-Hernández, A. ScratchR : a platform for sharing user-generated programmable media. 2007. http://dspace.mit.edu/handle/1721.1/42977?show=full.
31. Raymond, E.S. The Cathedral and the Bazaar. O'Reilly Associates, Inc., 1999.
32. Reas, C. and Fry, B. Processing: programming for the media arts. *AI & Society* 20, 4 (2006), 526-538.
33. Wash, R. and Rader, E. Public bookmarks and private benefits: An analysis of incentives in social computing. *Proc. American Society for Information Science and Technology* 44, 1 (2007).
34. Wenger, E. Communities of practice. http://www.ewenger.com/theory/index.htm.
35. Yee, K. Fisher, D. Dhamija, R. and Hearst, M. Animated Exploration of Dynamic Graphs with Radial Layout. *Proc. IEEE Symposium on Information Visualization 2001*, IEEE Computer Society (2001), 43.