Creative Networks:
Socio-Technical Systems for Loosely Bound Cooperation

Yannick Mahugnon Assogba
B.Sc. Computer Science (Hons)
Concordia University, Montreal June 2006

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
OCT 2 6 2009
LIBRARIES

ARCHIVES

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in Partial Fulfillment of the Requirements for the Degree of Master of Science in Media Arts and Sciences at the Massachusetts Institute of Technology.

September 2009

© 2009 Massachusetts Institute of Technology. All rights reserved

AUTHOR

Yannick Assogba
Program in Media Arts and Sciences
August 7, 2009

CERTIFIED

Judith Donath
Director of Sociable Media Group, MIT Media Lab
Thesis Supervisor

ACCEPTED

Deb Roy
Chair, Department Committee on Graduate Studies
Program in Media Arts and Sciences

Creative Networks:
Socio-Technical Systems for Loosely Bound Cooperation

Yannick Mahugnon Assogba
B.Sc. Computer Science (Hons)
Concordia University, Montreal June 2006

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in Partial Fulfillment of the Requirements for the Degree of Master of Science in Media Arts and Sciences at the Massachusetts Institute of Technology.

September 2009

© 2009 Massachusetts Institute of Technology. All rights reserved

Abstract

This thesis introduces a programming environment entitled *Share* that is designed to support and encourage *loosely bound cooperation* between individuals within communities of practice through the sharing of code. Loosely bound cooperation refers to the opportunity members of communities 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.

Our hypothesis is that providing fine-grained, publicly visible attribution of code sharing activity within a community can provide socially motivated encouragement for participation as well as pragmatic value of being able to better track downstream use and changes to contributions that an individual makes.

We shall present a discussion of loosely bound collaborative practice in various creative domains and the technological and social factors that contribute to the salience of these forms of cooperation today as well as discussing the motivational factors associated with open source development and how they differ in the case of cooperating individuals who do not share a project. We will also 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.

SUPERVISOR	Judith Donath
TITLE	Director of Sociable Media Group, MIT Media Lab

Yannick Assogba

Creative Networks:
Socio-Technical Systems for Loosely Bound Cooperation

THESIS READER

Mitchel Resnick
LEGO Papert Professor of Media Arts and Sciences
Program in Media Arts and Sciences, MIT

Yannick Assogba

Creative Networks:
Socio-Technical Systems for Loosely Bound Cooperation

THESIS READER

David P. Reed
Adjunct Professor
Program in Media Arts and Sciences, MIT

Acknowledgements

Above all my thanks go to God, for bringing me to this place and through this process, for peace and provision, inspiration and ability, for me, this work would not have been possible without Him. I would also like to thank my advisor Judith Donath, for education and encouragement, guidance and critique, freedom and fun—I have grown greatly as a researcher and critical thinker in my time here. My appreciation also goes out to the rest of the Sociable Media Group that I had the pleasure of knowing and working with, to Aaron, Alex, Drew and Dietmar, thanks for providing a wonderful and diverse intellectual environment and fun times in and out of the lab. To other friends and colleagues around and beyond the lab, Emily, Kate, Eric, Seth, Siggi, Simon, Ana-Luisa, Doug, Sajid, Ken, and so many others, thanks for your company, help, and general awesomeness.

I would like also to express my gratitude to my thesis readers David Reed and Mitchel Resnick for their feedback, perspective, and careful eyes. My appreciation also goes to the many members of the Media Lab that I have had the pleasure of learning from; it has been a pleasure being in such a diverse and passion-filled environment. Also, this thesis draws on ideas from so many others, almost all strangers, most are listed in the bibliography but I would like to thank you all here too.

Thanks to Jason Lewis + Family, for friendship, advice, encouraging me to come here and for being like family to me when I'm so far from my own.

Finally to my family, thanks for your love, support, and prayers. My love and sincerest appreciation go to you. This thesis is dedicated to my mother Lucile, my greatest source of inspiration when I feel weary.

Table of Contents

Abstract 3

Acknowledgements 9

Introduction 13

Design Proposal & Hypothesis 14

Why Art? 15

Outline 16

Sampling & Collaborative Practice in Cultural Production: An Art-Historical Perspective 18

Artist Collectives 19

The Impressionists 19

De Stijl 22

Sampling 24

Literature 25

Music 26

DJ Culture & Intertextuality in Hip-Hop 27

Creativity & Copyright Law: Tensions of ownership 32

Fallout 34

Plunderphonics 34

YouTube 35

What about Fair Use? 36

Copyleft 39

New Economies: The Sharing Economy & Commons Based Peer Production 40

Political Affordances of Network Technology 41

Reward & Motivation in Free and Open Source Software 42

Loosely Bound Cooperation 44

Development Across Project Boundaries 45

Challenges Across Project Boundaries 46

Processing, The Community; Forums and Other Existing Tools 48

Related Work 48
Scratch 52
OPENSTUDIO 54
GitHub 55

Design & Implementation 56
Share Server 60
Share Client 63
File Browser 64
Code Search 66
Editor 67
Permissions 70
Explicit References 70
Comments 71
Bookmarks 72
Synchronization 74
The Network Browser 75
Runtime 79
Security Concerns 79

The Share Experiment: Results & Discussion 82
Experiment Design 83
Recruitment and Participant Demographics 84
Prior Code Sharing Experience 86
Experience Using Share 90
Quantitative data 90
Explicit (@saw) References 95
Survey Response 96

Conclusion & Future Work 104
Future Work 107
Design Opportunities 107
Research Questions 108

Appendix A 110

Bibliography 118

Chapter 1. **Introduction**

"Bernard of Chartres used to say that we are like dwarfs on the shoulders of giants, so that we can see more than they, and things at a greater distance, not by virtue of any sharpness of sight on our part, or any physical distinction, but because we are carried high and raised up by their giant size."

—John of Salisbury (1159)

This thesis concerns itself with new opportunities available to creative individuals to locate themselves in larger and larger communities mediated by networked technology and the opportunity that presents for greater interplay between the individual and socio-contextual aspects of creative endeavor. Over the last several years we have seen the growth in ability of digital communication networks to organize the actions of large numbers of distributed individuals in performing work at a scale more traditionally associated with that of companies and corporations. 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 (Kanefsky, Barlow & Gulick, 2001) or the ESP game (von Ahn & Dabbish, 2004) are but a few examples of how the work of individuals collaborating at immense scale is synthesized to produce something that is in a sense 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 create. 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 allows individuals to pursue *independent* goals yet still be able to take advantage of the properties of the network to *help each other along the way.* We refer to this form of collaboration as *loosely-bound cooperation.*

Central to this thesis is the idea that individual creativity is borne out of a rich heritage of existing work and a context that includes the creative output of the communities to which we belong. Fischer et al. (2005) argue that while we typically hold an image of the creative individual as a "lone thinker" laboring at his art in isolation until it is perfected and ready to be shown to the world, this is not the whole picture; we must take into account the intellectual context in which our creative talent operates in considering what makes us such capable and interesting creatures. This intellectual context is inherently a social one, providing feedback, validation and raw material from which new creative output is synthesized. This is not to discount or belittle the unique contribution brought by the individual, but rather suggests that the relationship between 'genius' (individual) and 'inspiration' (socio-contextual) is a complementary one, each emerging from the other, and that there is opportunity to build tools that support this relationship.

Design Proposal & Hypothesis

We propose a novel programming environment geared towards supporting loosely-bound cooperation between programmers within communities of practice; our prototype is initially targeted at programmers using the *processing* programming language (Reas & Fry, 2007), a language geared towards multimedia artists, designers and others interested in using code as a central part of their practice. Our tool does this by *sharing* all the code written in it with all other members of the community and also *tracking* its reuse, providing *fine grained attribution* of where code came from as well *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 will provide an interactive visualization of the entities within the system (users and code artifacts) and the relationships between them. The visualization allows users to navigate the social context around code artifacts.

Explicit reference and linking of artifacts. The system will provide 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).

Our *hypothesis* is that we can leverage *public display of attribution to provide reward for, and motivate participation in,* code-sharing based cooperation between individuals who are in pursuit of independent goals. The central questions that we are asking with respect to our design include:

1. What rewards does the visualization of attribution provide to the original contributor? Do these rewards lower the barrier towards openly sharing ones code?
2. Are individuals able to track the re-appropriation of code they have contributed? If so what are the benefits to doing so?
3. Does working in such a system disrupt their regular work practice? Can users program without being encumbered by the notion of participating in a community?

Why Art?

So why situate a study like this one within the arts? We do so because we believe artistic practice provides strong examples of individuals working towards their own goals without a shared task or goal, yet often sharing a context in which that work is produced. We believe that the strong relationship between artists and the code they produce makes this a suitable test bed to investigate these issues around motivating loosely bound cooperation.

Outline

This thesis can roughly be divided into two parts, in the first part we present a broad overview of the history of sampling and collaborative practice within the realm of art production and the effect that technology has had on the nature of those practices. We then take a look at the tensions between contemporary sampling and remix practice and modern copyright and intellectual property law. We also discuss the emergence of new frameworks for exchange that better support these collaborative practices, particularly looking at the salience of 'sharing economies' and new models for non-market production and distribution that are enabled by modern communications technologies. Finally we survey some of the reasons individuals choose to participate in these new economies and the benefits and trade-offs in doing so, we will also further unpack what we mean by loosely-bound cooperation by contrasting it with the model of mass collaboration that is enabled by similar technologies.

The second part of the thesis starts with a look at a set of existing projects and tools that serve as exemplars for the kind of system we propose to build, we shall then describe the design and functionality of the prototype that we created and discuss our initial user study and the feedback received from our users. We then conclude the thesis and look at some future directions that the current research opens up.

Chapter 2. **Sampling & Collaborative Practice in Cultural Production: An Art-Historical Perspective**

"Immature poets imitate; mature poets steal; bad poets deface what they take, and good poets make it into something better, or at least something different." — T.S. Elliot

This chapter aims to provide a brief discussion on the history of sampling and other collaborative practices in the arts; we do this for a number of reasons. Firstly to advance an argument that as a means of artistic and cultural production its results are not inferior to what we may conceive of as 'original'. Secondly to see that it has historically been part of the practice of many artists and hopefully diminishing conceptions we may have that the product of art and more generally creativity is the product of focused individual genius bereft of the context (particularly social) that played a part in its creation. Thirdly to provide a basis to discuss the structural changes brought about within these practices from the proliferation of cheap, fast digital communication networks. We will make brief excursions into three main areas, Visual Arts, Literature and Music to draw examples that relate to sampling and collective practice over the last century or so.

Artist Collectives

The visual arts provide great examples of collaborative practice amongst artists, with numerous, schools, movements and looser collectives and associations. We draw comparison between these 'schools' and the notion of *communities of practice* advanced by Lave and Wenger (1991), which describes a group of individuals with a shared set of practices or tools working within a community. Our discussion here focuses on the Impressionist and De Stijl movements.

The Impressionists

The first impressionists were a loosely affiliated group of artists working around Paris in the 1860's. In "*Impressionists Side by Side*" art historian Barbara Ehrlich White (1996) documents the "friendships, rivalries and artistic exchanges" of some of the most celebrated impressionist painters, including Degas, Monet, Cezanne and Manet. In the book White examines their relationships in terms of overlapping pairs of working relationships. Their collaborative practices included working in each other's presence, painting the same subject (literally painting side by side), sometimes copying and in rare instances correcting each other's work[1], as well as overt collaboration on projects. Indeed some critics of the day suggested their art was too similar, however White insists that "the Impressionists strove for individuality" asserting (and demonstrating in the book) that "Each had different ideas, approaches, attitudes, and contributions". The relationships and collaborative practice of the artists allowed them to develop their craft and progress the art form by learning from each other. In the case of the female artists, the relationships with male counterparts allowed them to receive attention to their work that would have been harder to garner given the nature of society at the time. Of their collaborations White says

> "It is not often realized how much these men and women relied on each other for camaraderie, support, inspiration, ideas and techniques. They not only learned from but also competed with each other and their art changed as a result. Without these friendly rivalries each artists work would not

1 Both Degas and Monet each corrected a painting by Cassatt and Morisot respectively, that the latter had sent to be exhibited (White, 1996 pp 3)

have been as rich ... Working together led them to important breakthroughs in style and theme; this resulted in artistic growth for all of the painters and in a great number of works of art." — (White, 1996)

Figure 1. Manet, *Gare Saint-Lazare,* d. 1873, Oil on canvas 36¾ × 45" (93 × 114 cm). National Gallery of Art; Gift of Horace Havemeyer in memory of his mother, Louisine W. Havemeyer. Source: White (1996).

Figure 2. Morisot, *On the Balcony,* c. 1871-72, Oil on canvas 23⅝ × 19⅝" (60 × 50 cm). Private Collection. Source: White (1996).

Figure 3. Manet, *Camille and Jean in the Garden (*or *The Monet Family in Their Garden at Argenteuil)*, 1874, Oil on canvas 24 × 39¼" (41 × 99.7 cm). The metropolitan Museum of Art, New York; Bequest of Joan Whitney Payson. Source: White (1996).

Figure 4. Renoir, *Portrait of Cammille and Jean in the Garden at Argenteuil,* 1874, Oil on canvas 19⅜ × 26¾" (49.2 × 67.9 cm). National Gallery of Art, Washington; Alisa Mellon Bruce Collection. Source: White (1996).

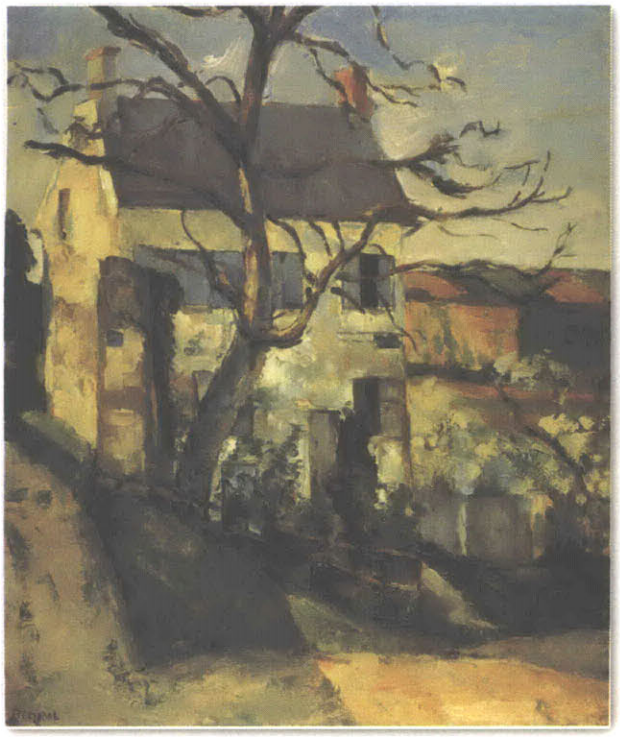

Figure 5. Cézanne, *House and Tree near the Read of the Hermitage, Pontoise,* 1874, Oil on canvas 25⅝ × 21¼" (66.2 × 53.3.9 cm). Private Collection. Source: White (1996).

Figure 6. Pissarro, *The Road of the Hermitage, Pontoise,* d. 1874, Oil on canvas 18 × 15" (45.7 × 38.1 cm). Private Collection. Source: White (1996).

De Stijl

De Stijl is the name of a loosely affiliated group of artists including painters (Mondrian, Van Doesburg) and architects (Oud, Rietveld, Wils), whose work was influential in the development of modern graphic design and international style architecture (Friedman, 1982). Formed in World War 1 Holland, the members of De Stijl did not all work in close physical proximity with each other, indeed many of them never met and barely knew each other personally (Overy, 1991 pp 7), communicating and sharing their work mainly through the monthly publication of a journal, "*De Stijl*" that gave the association its name. However they shared a common philosophy as to the form and function of their art and their work is characterized by their shared palette of straight lines, right angles, the use of primary colors red, blue and yellow (in addition to the 'non-colors' white, black and grey). Their collaboration demonstrates the movement and development of ideas even across genres.

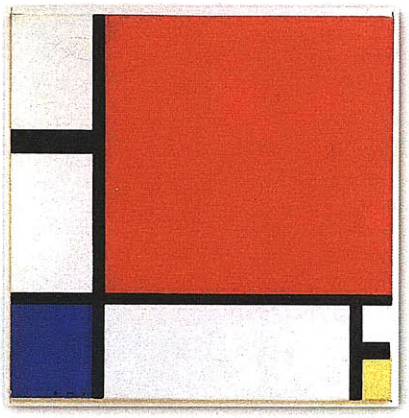

Figure 7 (above). Mondrian P. *Composition with Red, Blue, and Yellow*, 1930, Oil on canvas 46 × 46 cm. Kunsthaus Zurich. Retrieved July 15, 2009 from Artstor. http://library.artstor.org/library/secure/ViewImages?id=%2FThWdC8hIywtPygxFTx5RnguXXopfFk%3D

Figure 8 (right). Rietveld G. *Red-Blue Chair*, design 1917-1918, c. 1974, deal wood, plywood, ebony aniline dye 40 × 20 × 27". The Minneapolis Institute of Arts; The Modernism Collection, Gift of Norwest Bank Minnesota. Retrieved July 15, 2009 from Artstor. http://library.artstor.org/library/secure/ViewImages?id=8D1Efjk2ODA0KyYrZD5%2FXnVHXnogd1t7fSY%3D

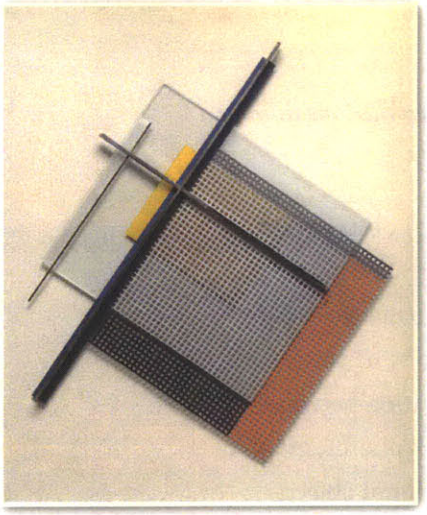

Figure 9 (far left). Domela C. *Construction,* 1929, glass, painted glass, painted metal, chromeplated brass, painted wood 89.9 × 75.2 × 4.2 cm. Collection Hirshborn Museum and Sculpture Garden, Smithsonian Institution. Source: Friedman (1982)

Figure 10 (left). Rietveld G. *End Table,* 1923, painted wood 61.6 × 48.9 × 48.9 cm. Collection Stedelijk Museum, Amsterdam. Source: Friedman (1982)

Figure 11 (left). Rietveld G. *Schröder house,* 1924–5, restored 1985–7. Source: Overy (1991)

There are countless other examples of the importance of the artists collective in the development of the artists and the art form, from the Pre-Raphaelite Brotherhood and Der Blaue Reiter to Dada and Fluxus. The notion of artist collaboration is certainly not new and likely not even controversial, it's importance however, is sometimes overlooked.

Sampling

A process in which a sound is taken directly from a recorded medium and transposed onto a new recording. (Fulford-Jones, 2009)

Among many others Manovich (2005, 2007) observes that cultural production in the internet age is increasingly comprised of practices that re-use, re-appropriate and re-contextualize existing media in the creation of new cultural content. In 2005 he stated "In most cultural fields today we have a clear-cut separation between libraries of elements designed to be sampled – stock photos, graphic backgrounds, music, software libraries – and the cultural objects that incorporate these elements", he then asks "Will the separation between libraries of samples and "authentic" cultural works blur in the future?" To look around now and answer that question one is forced to say, YES (though our legal regulations are still catching up); across diverse media from image and moving image to music, everything is up for grabs in the construction of new works, Manovich's metaphor of 'database' (Manovich, 2001) previously applied to film and new-media can be increasingly applied to all forms of cultural production, high and low

Sampling as a mechanism for collaborative practice holds particular salience with respect to this thesis as it is the method of collaboration most directly supported by our tool. As a method it has a long history, likely stretching as far back as the means for fixed storage and reproduction of material, and certainly well before the 'Internet age'. While we could draw even more examples from the visual arts to demonstrate this, particularly when we look at collage-based work. We will use examples from literature to show its antiquity, and examples from contemporary music to examine how technology changes the scales (across time and space) at which samples are transmitted and reused

Literature

"Literature has always been a crucible in which familiar themes are continually recast" — *Michael Maar*

Author Jonathan Lethem's essay *The Ecstasy of Influence: A Plagiarism Mosaic* (Lethem, 2007) begins with an account that almost makes Victor Nabokov to be a plagiarist, describing the plot of a manuscript by German author Heinz von Lichberg entitled *Lolita* that bears striking resemblance to its more well known namesake, and which had been published forty years earlier. Michael Maar, author of a book exploring the relationship between the two works, suggests that it is however not appropriate to consider Nabokov a plagiarist, suggesting that the works differ such that *"Nothing of what we admire in [Nabokov's] Lolita is already to be found in the tale; the former is in no way deducible from the latter"* (Maar, 2005).

Lethem's essay makes a case, historical, contemporary and visceral (many of the words in his essay are in fact lifted from others—he gives a key at the end of the essay), as to how natural acts of appropriation like these are. He writes

> "Most artists are brought to their vocation when their own nascent gifts are awakened by the work of a master. That is to say, most artists are converted to art by art itself. Finding one's voice isn't just an emptying and purifying oneself of the words of others but an adopting and embracing of filiations, communities, and discourses. Inspiration could be called inhaling the memory of an act never experienced. Invention, it must be humbly admitted, does not consist in creating out of void but out of chaos. Any artist knows these truths, no matter how deeply he or she submerges that knowing." (Lethem, 2007)

When we look at 16th century European literature it seems that it was common and accepted practice for works of literature to borrow, ideas, plots and phrases from other works of the day and of antiquity without reference or attribution. Since at least the eighteenth century, scholars have been investigating the sources for Shakespeare's plots and have had an understanding that *most* of the

underlying stories for his poetry were not created by him (Metz, 1989, pp xi). Indeed Geoffrey Bullough (1901-1982), Professor of English Language and Literature, Kings College, Cambridge has published an 8 volume series on Shakespeare's sources (Bullough, 1957). At least three examples of the "pound of flesh" demanded by Shylock are found in works prior to *The Merchant of Venice,* (*The Jew, Zelauto* and *Il Pecorone* (Muir, 1978 pp 86–90)). The story of *Romeo and Juliet* exists prior to Shakespeare's version as an English translation of a French translation of an Italian adaptation of another Italian story, the story remaining fairly consistent between all the versions, with various characters being added along the way. Scholars suggest that Shakespeare's main source was likely the 1562 poem, *The Tragicall Historye of Romeus and Juliet* by Arthur Brooke (Muir, 1978 pp 39).

Literary criticism has recognized this practice using, among others, the term *intertextuality*, the shaping of one text by another, (Kristeva, 1986), such as that visible between James Joyce's *Ulysses* and Homer's *Odyssey* (Attridge, 2004 pp 122). We see in these examples masterful use of existing cultural content in the production of something fresh and new. We do not seek to diminish the immense creativity brought to the table by individuals who created these works; we merely seek to highlight that it was in their common practice to s *sample* when necessary to complete their work.

Music

Give me two turntables and I'll make you a universe. — DJ Spooky That Subliminal Kid

One art form that is arguably in the forefront of discussions on issues and rights with respect to the practice of sampling to create new works is music. Enabled by digital representations and tools that transform sound itself into a malleable raw material to be used in the creation of new work, audio culture has certainly been affected by sampling practice. Sound art and music also provide clear examples of how technology has allowed collaborative practice to transcend geography

In 1937 composer John Cage writes.

"I believe that the use of noise to make music will continue

> and increase until we reach a music produced through the aid of **electrical instruments which will make available for musical purposes any and all sounds that can be heard.** Photoelectric, film, and mechanical mediums for the synthetic production of music will be explored. Whereas, in the past, the point of disagreement has been between dissonance and consonance, it will be, in the immediate future, between noise and so-called musical sounds. The present methods of writing music, principally those which employ harmony and its reference to particular steps in the field of sound, will be inadequate for the **composer who will be faced with the entire field of sound.**" [Emphasis Added] (Cox, Warner, 2004 pp 25)

Cage sought to open up the palette of sounds available in the making of music. Cage and futurists such as Luigi Russolo[2] problematize the term "Music" by using the word "noise" to refer to sounds in a general way without the restrictions enforced by the musical community of his time (and still present in music departments today). He saw a future, enabled by a new set of tools, in which all sound would be available for use in the creation of the new music. We are indeed faced with that time now, digital tools easily allow any sound to be captured, manipulated and re-appropriated in the creation of new work.

DJ Culture & Intertextuality in Hip-Hop

Prefigured by Laszlo Moholy-Nagy's exhortation to transform the phonograph into a tool for *production* as opposed to merely ***re-production*** (Cox, Warner 2007 pp. 329), DJ Culture is the modern configuration of the futurists' vision translated into the vernacular. It is exemplified by, yet by no means exclusive to, Hip-Hop, a genre whose origins lie with a group of individuals, DJ's, such as Kool Herc, GrandMaster Flash and Afrika Bambaataa, who were isolating breaks and beats from existing records and transforming

2 Luigi Russolo was an Italian futurist who wrote a seminal manifesto entitled "The Art of Noises" in 1913 in which he called for the orchestra of the future to incorporate the entire palette of noise-sounds that were emerging at the time. Reacting to the evolving auditory ecology of Europe after the industrial revolution that was filled with new exciting sounds of machinery, he saw an opportunity and an imperative to incorporate these new sounds in the service of culture.

them through cuts, scratches and uncanny combinations into completely new sounds and rhythms. Hip-Hop has always been at home with the notion of re-appropriation, with DJ's, rappers and producers borrowing from and reforming the past into a new present. Paul Miller aka DJ Spooky That Subliminal Kid writes. "Each and every source sample is fragmented and bereft of prior meaning — kind of like a future without a past. The samples are given meaning only when re-presented in the assemblage of the mix." (Cox, Warner, 2004, pp 349–350). DJ's are able to channel the collective consciousness borne from our familiarity with the products of mass culture and use those connections to create something new in the light of the past; this play with the familiar to create the unfamiliar (in other words new) is a definite boon of remix culture.

The willingness to quote extends from the music into the lyrics of hip-hop; artists will often quote the lyrics of others directly in their songs, particularly from well-known songs or artists. This is an act that must be done with great skill, as, ironic as it may seem — though it isn't really when you understand it, originality is key in proving oneself a competent rapper. In quoting another's lyrics one must do so cleverly and creatively to avoid being called out as a 'biter' (plagiarist). The danger of this is high as the quote is likely to be recognized by others, indeed that is the point, and if not done deftly risks failure. Jay-Z, one of Hip-Hops more successful artists, defends his quoting from accusations of "biting" leveled against him. On the track *What More Can I Say* (Carter, 2003, track 3), he rhymes

> "I'm not a BITER / I'm a WRITER for myself and others / I say a BIG verse / I'm only BIGGIN' up my brother / BIGGIN' up my borough / I'm BIG enough to do it"

The emphasis on the word "BIG" is a response to accusations that Jay-Z uses much too material from deceased rapper Notorious BIG. While an accepted and common practice, there are certainly tensions over quoting in rap, these tensions however aim to keep it honest. Most of Jay-Z's quoting has come from well known tracks and are generally recognized (at least by fans) as being respectful of their original, they are also likely tolerated because of his ability to skillfully weave them into new songs

The entire chorus of rapper Cassidy's hit single *I'm a hustla* (Reese, 2005, track 2) is a quote from a well-known Jay-Z song. In

fact rather than rap it himself, Cassidy uses a Jay-Z's recording of the words *"I'm a hustler homie"* to construct his chorus. This use is not viewed as plagiarism but clever re-appropriation. In the track he goes on to echo another Jay-Z lyric, *"you made it a hot line, I made it a hot song"* referring to the process of taking a single line from one track and building a whole new one around it. It is indeed a meta-reference as the original use of this phrase is in fact by Jay-Z in another song "*Takeover*" (Carter, 2001, track 2), referring to Jay-Z's borrowing of a line from a song by artist *Nas* and making it the chorus of one of his songs[3]. There is certainly an intricacy in how these quotes are used and re-used that adds a certain richness and depth to the form.

Some critics of remix culture argue that 'originality' is always superior to remix, which they often view as a form of inferior copy. This argument I believe comes from a somewhat facile understanding of what remix culture is actually able to produce and the mode in which it operates, it may also come simply from exposure to bad remix. Whether it is a track by *Girl Talk,* a DJ who on one album combines over 300 samples in under 50 minutes of music — see fig. 12, or masterful turntablists such as the X-Ecutioners, who can take a few seconds of material and cut and scratch it into something so new the old is barely recognizable. One can look at these as well as other works such as *the grey album* (a mashup of the Beatles *White Album* and Jay-Z's *Black Album*) for evidence of highly skilled and creative work emerging from the remix culture.

Just as the futurists sought to take the new sonic material of the industrial revolution present in their surroundings in the creation of new music; DJ culture, remix culture, seeks to make use the cultural content we are bombarded with (or to look at another way, have tremendous access to) in the creation of art and *new* cultural content.

When we take a look at the technology for creation and distribution of musical work today we see an even looser configuration of the artists that are involved in each other's practice. The looseness of association between members of a movement such as De Stijl does not compare with the diversity of sources, across genres and time periods available to the modern musician with a sampler and an Internet connected computer.

3 On the song "Takeover" (2001) Jay-Z is referring to taking the lyric "I'm out for presidents to represent me" from Nas' song "The world is yours" (1994) and turning it into the chorus for one of his songs "Dead Presidents 2" (1995)

> "This morning I was listening to a Thai lady singing; I can hear the sound of the St Sophia Church in Belgrade or Max's Kansas City in my own apartment ... the whole global musical culture is also available. That means that a composer is really in the position, if he listens to records a lot, of having a culture unbounded, both temporally and geographically, and therefore it's not at all surprising that composers should have ceased writing in a European classical tradition, and have branched out into all sorts of experiments" — Brian Eno (Cox, Warner 2004, pp 128)

The above quote while liberating in one sense also underscores the very real possibility of an artist today hearing her work incorporated into the work of another artist with which she has no previous social connection or contact.

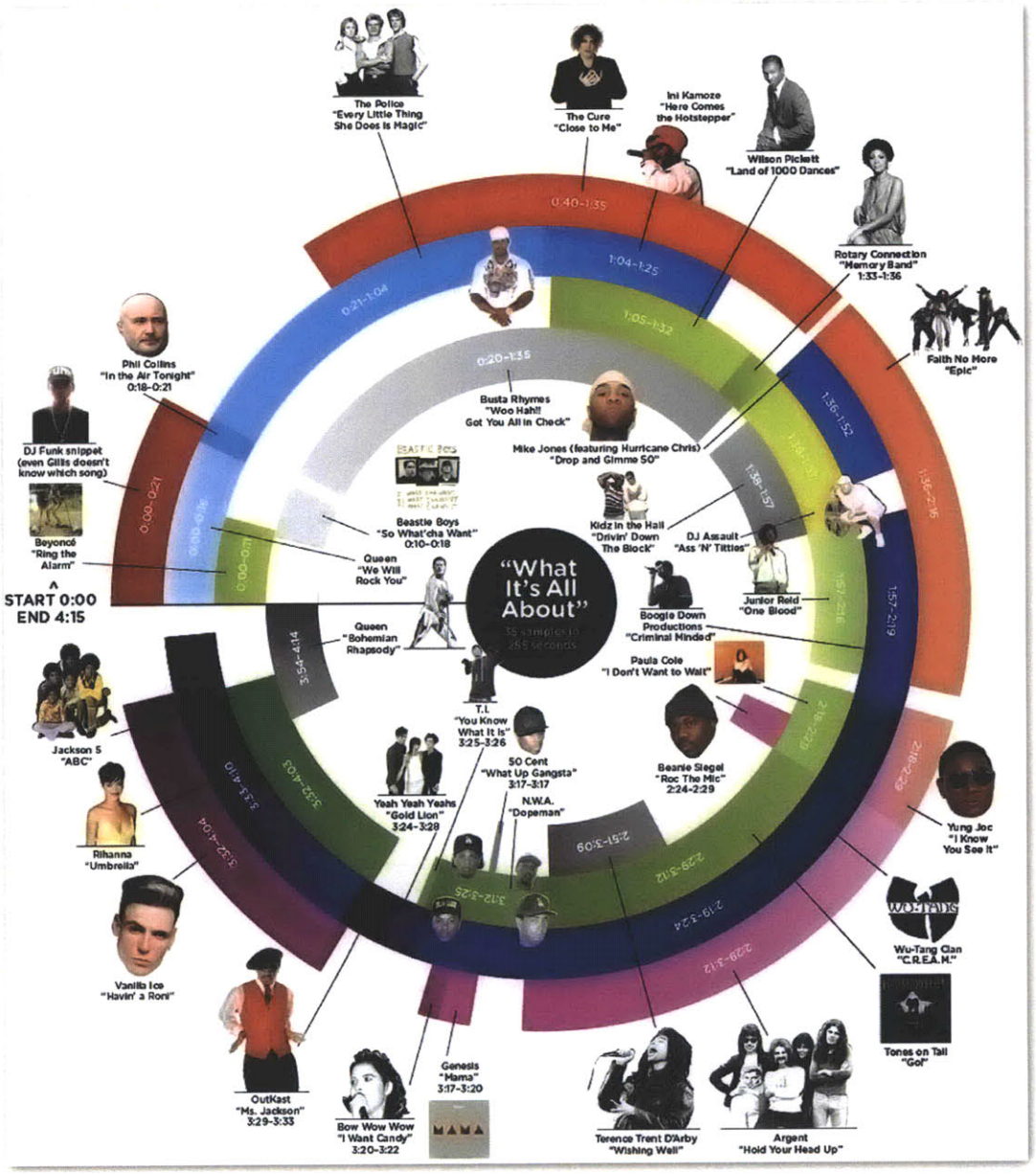

Figure 12. *Visualisation of Samples used in Girl Talk Single, "What It's All About"*, Wired Magazine 2009, Retrieved on July 15 2009 from Wired.com, http://www.wired.com/special_multimedia/2008/pl_music_1609

Chapter 3. **Creativity & Copyright Law: Tensions of ownership**

When "the same thing" is so different that it constitutes a new thing, it isn't "the same thing" anymore — Chris Cutler

While this thesis is not primarily about copyright law, we do recognize that law shapes our conceptions, as individuals and a society, with regards to these issues of sampling, remix, originality and ownership. In this chapter we aim to provide a brief lay of the land with respect to modern US copyright law and its tensions with remix culture. This discussion will also serve as a launching pad to look at alternatives that have emerged over the last twenty or so years.

The goal of copyright, to support and encourage invention and creativity for public benefit, is one that we share in the design of our system. Structural changes brought about by technical advancements in communications technology however, create a tension between existing models of copyright and new collaborative practices and emerging mores around cultural reuse and appropriation.

U.S. copyright law provides exclusive rights to the creators of 'original works of authorship' that allows creators to control, among other things, the reproduction, the creation of derivative works, and the public performance of their work ("U.S. Copyright Office - Copyright Law", n.d.). Copyright law however places a limit on the

amount of time a work can be controlled by its author (or 'rights holder' in cases where the author sold his/her 'right'). The rationale for this being that while it is in the public interest to allow the creator the opportunity to derive compensation from their work, and therefore be encouraged to produce more, yet it would not be in the public interest to allow the copyright holder (or their estate) to have a monopoly on their work indefinitely (Benkler, 2007 pp 36–37). This should be evident when we consider the nature of information production, particularly that new information is created from old information, thus limits on copyright terms allow new information to be developed from old by a wider pool of people after the grace period has expired. This limit was originally 14 years, with an opportunity for the author to obtain another 14 years of protection if they are still alive, but has been extended over time and now lasts 70 years plus the life of the author (or a maximum of 120 years for works for hire or other anonymous sources) ("United States Copyright Office A Brief Introduction and History", 2009).

Lawrence Lessig, author and law professor, argues in his book *Remix* that copyright in its current form supports a *Read-Only (RO)* culture (borne out of print and other technologies for mechanical reproduction) that today criminalizes or makes prohibitively expensive the re-appropriation of existing cultural content for use in new contexts (Lessig, 2008). In this culture ownership is absolute and control is king. He argues that new technologies encode a more *Read-Write (RW)* culture and it is not in the public interest to relentlessly maintain current practice around certain cases of copyright infringement, particularly with regard to remix and re-appropriation. The tensions around who owns the cultural products of artists and other creative individuals, and what 'consumers' are allowed to do with it force us to reconsider what it is we want out of our intellectual property frameworks.

Our view is that prevailing technological shifts have changed how information is consumed and produced and create new opportunities for creativity and invention that work on the edge (and sometimes outside of) current copyright frameworks, this is a view shared by others and there has been a movement to create alternative frameworks that better support RW creative practices. Before looking at these alternative frameworks, we shall look at a few examples of how read-write culture clashes with the rights of copyright holders.

Plunderphonics

Plunderphonics is a term coined by Canadian composer John Oswald to describe the exclusive use of sound samples in the creation of a new audio work, in particular he alludes in his essay *"Plunderphonics, or Audio Piracy as a Compositional Prerogative"* that in plunderphonics it is important that the source material is recognizable (Oswald, 1985). *Plunderphonic* is also the title of an EP that Oswald released featuring tracks that used source material from artists such as the Beatles, Michael Jackson, Dolly Parton and Elvis Presley, attribution was made clear yet permission for their use was never sought. Most tracks on the EP are made entirely from the work of a single artist (and most often a single work), transformed mainly by changes in playback speed and chopping up and rearranging pieces of sound. Four months after release, upon threat of litigation, Oswald ceased distribution of the album and surrendered his remaining copies to the Canadian Recording Industry Association to be destroyed (Oswald 1990). While the copyright holders were certainly within their 'rights' to stop Oswald, it is important that we ask ourselves what the possible reasons they may have for doing so, and who the potential losers and winners of this action are. The first reason that comes to mind may be economic, on the face of it seems unfair for someone to be making money of the back of someone else's work, in this case however Oswald never offered the work for sale, copies were distributed to radio stations and libraries and listeners encouraged to dub them to tape. We are also left with the option that artists may simply not want their work to be used in particular ways (i.e. they may not like the outcome), however is this the purpose of copyright? Should it be? Do Oswald's Plunderphonics discourage or de-incentivize the original creators and thus harm the public good? These answers to these questions are not clear-cut or universal across use cases or individuals on either side of the equation, especially when using current copyright decisions as a guiding model.

YouTube

Stephanie Lenz is a mother who put a thirty second video of her thirteen month old son dancing to the song "Let's go Crazy" by artist Prince on YouTube. This was not a professionally recorded video, but one by a mother who grabbed her camcorder to record a moment in her young child's life and aimed to share it with friends and family (Lessig, 2008 pp 1-4). This re-appropriation of culture was more accidental than artistic, but it still resulted in her being accused of copyright infringement. About four months later YouTube sent her a notice informing her that her video would no longer be viewable because of a complaint from the Copyright holder of Prince's song, Universal Music Group. While Lenz was able to countersue Universal Music Group and had a judge declare that her use was indeed "fair use" ("Lenz vs. Universal | Electronic Frontier Foundation", n.d.), this case demonstrates tensions of ownership in works that use pre-existing media; does Universal have any rights to ownership over Mrs. Lenz and her sons' experience?

While the above is an extreme example it points to the extraordinary lengths corporate interests are willing to go to protect their artists' work, though in this case you would ask protect it from what? This makes the situation even more dire/impossible for those who seek to mix and 'mash up' culture intentionally. One of the more visible places this tension is playing out is YouTube, a search for "Mashup" on YouTube will yield hundreds of results. "Mashups" are a form of media that combine the audio or visuals from disparate sources into a new video (or song in the case of strictly audio mashups). One particular genre of mashup is the *anime music video*; these consist of typically of video from one or more anime shows edited to correspond with an audio soundtrack usually unrelated to the anime being used ("Anime Music Videos", 2006). These videos would likely be considered illegal in U.S. copyright law but it is hard to see how they compete in the same domain with the works from which they borrow material (the anime or the audio track), and thus threaten the livelihood of those creators. Current YouTube policy is to "silence"[4] these videos when a complaint of copyright infringement is received.

4 The audio track is muted while the video is allowed to play, a notice is displayed to explain why there is no audio. An example can be seen here www.youtube.com/watch?v=p6ldtvX6xsM

While these technologies are not that new, their current availability, and importantly their new distribution channels, makes it such that anyone can repurpose cultural artifacts, anyone (and their mother) can be a DJ. The changes enabled by technological innovation are an important theme underlying this thesis and result in an increasing amount of friction with current law as these practices become more widespread, connected and sophisticated.

What about Fair Use?

Those familiar with U.S. copyright law are aware of the limitations on its application, including a limit to protect "fair use" of copyrighted works. However what counts as fair use is difficult to categorize, it is important to remember that in this context "fair use" is a legal term with specific meaning that may or my not overlap with your conceptions of a "fair" use of a copyrighted work. The U.S. copyright office describes fair use in the following manner.

> "Section 107 contains a list of the various purposes for which the reproduction of a particular work may be considered fair, such as criticism, comment, news reporting, teaching, scholarship, and research. Section 107 also sets out four factors to be considered in determining whether or not a particular use is fair:
>
> 1. The purpose and character of the use, including whether such use is of commercial nature or is for nonprofit educational purposes
>
> 2. The nature of the copyrighted work
>
> 3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
>
> 4. The effect of the use upon the potential market for, or value of, the copyrighted work
>
> The distinction between fair use and infringement may be unclear and not easily defined. There is no specific number of

words, lines, or notes that may safely be taken without permission." ("U.S. Copyright Office - Fair Use", 2009)

As indicated above "fair use" has a very limited and often difficult to interpret scope primarily protecting education and research as well as criticism and parody (although its use has been extended to include things like time-shifting media (Sony Corp. of America v. Universal City Studios, Inc. 464 U.S. 417, 1984)) Determined on a case-by-case basis, an individual accused of copyright infringement takes on significant legal burden and risk when using a fair use defense; this is true whether you are an amateur (like Stephanie Lenz) or a professional (like John Oswald). The myriad factors involved in determining fair use make it somewhat of a moving target (a potentially expensive moving target should you be on the wrong end of it). The only way to really examine the scope of fair use is to examine previous cases; the following examples are from the Stanford Copyright and Fair Use Center.

"***Fair use.*** The rap group 2 Live Crew borrowed the opening musical tag and the words (but not the melody) from the first line of the song "Pretty Woman" ("Oh, pretty woman, walking down the street "). The rest of the lyrics and the music were different. ***Important factors:*** The group's use was transformative and borrowed only a small portion of the "Pretty Woman" song. The 2 Live Crew version was essentially a different piece of music and the only similarity was a brief musical opening part and the opening line. (Note: The rap group had initially sought to pay for the right to use portions of the song but were rebuffed by the publisher who did not want "Pretty Woman" used in a rap song.) (Campbell v. Acuff-Rose Music, 510 U.S. 569 (1994).)

Not a fair use. The artist, Jeff Koons, created a series of porcelain sculptures based upon a photograph of a man and woman holding puppies. Although certain aspects were exaggerated, the photo was copied in detail. Koons earned several hundred thousand dollars from sales of the sculptures. ***Important factors:*** Although Koons claimed fair use under a parody theory the sculptures were part of his "banality"™ series the court disagreed claiming that the sculptures did not parody

the work. The court also noted that it did not matter whether the photographer had considered making sculptures; what mattered was that a potential market for sculptures of the photograph existed. (Rogers v. Koons, 960 F.2d 301 (2d Cir. 1992).)

Fair use. A person running for political office used 15 seconds of his opponent's campaign song in a political ad. ***Important factors:*** A small portion of the song was used and the purpose was for purposes of political debate. (Keep Thomson Governor Comm. v. Citizens for Gallen Comm., 457 F. Supp. 957 (D. N.H. 1978).)

Not a fair use. A poster of a "church quilt" was used in the background of a television series for 27 seconds. ***Important factors:*** The court was influenced by the prominence of the poster, its thematic importance for the set decoration of a church and the fact that it was a conventional practice to license such works for use in television programs. (Ringgold v. Black Entertainment Television, Inc., 126 F.3d 70 (2d Cir. 1997).)"

("Stanford Copyright & Fair Use - Summaries of Fair Use Cases", 2007)

We can see from the definition and the examples above that fair use is not intended to cover derivative works of all kinds. While it recognizes the importance of being able to quote for the purposes of education, critique and commentary (including art which functions as social commentary) and considers the transformative (vs. purely reproductive) nature of the work as well as considering whether the new work can substitute for the old or the new works effect on the commercial viability, it does not cover many, many forms or remix and creative re-appropriation (for example it has not generally protected the sampling prevalent in hip-hop and other popular music genres — in these domains samples need to be licensed — nor does it seem to protect YouTube from litigation on the audio tracks of Anime Music Videos). Our next section looks at alternative frameworks that seek to create spaces more amenable to sampling and remix practice.

Copyleft

Emerging out of the tensions between remix cultures and current copyright law are movements that seek to provide alternative frameworks, legal and cultural, to those of modern copyright. The *copyleft* movement is a framework that emerged from a part of the software community that desired to create an ecosystem that encourages openness, sharing and the creation of derivative works. Copyleft licenses, such as the GNU foundations *General Public License* ("GNU General Public License", 2007) or the *Creative Commons* licenses ("Licenses - Creative Commons", 2009), typically relinquish a number of the rights provided to authors under copyright (which is automatic in the U.S.), while sometimes adding a set of alternate restrictions on use. Rights relinquished often include control over reproduction and the creation of derivative works, they are thus freely available for re-appropriation. Restrictions added often, but not always, seek to require that derivative works are released under a similar copyleft license thus expanding the pool of work in the commons; they also often require some form of attribution to the contributor of the work being reused. Some, but not all copyleft licenses restrict commercial uses, it should be noted that many copyleft proponents do not think the freedoms their licenses provide preclude commercial use; this is well encapsulated in the phrase *"free as in free speech, not as in free beer"* ("The Free Software Definition", 1996). It is within these frameworks that we situate our work. Our proposal is a design that seeks to better support those already participating within these alternative frameworks for intellectual property control. The following chapter will examine the modes of production within these alternate frameworks in a bit more depth with particular focus on free and open source software.

Chapter 4. New Economies: The Sharing Economy & Commons Based Peer Production

In this chapter we seek to take a broader look at the alternative frameworks for creative and cultural production that we introduced earlier, particularly looking at how they are enabled by the ubiquity of networked technologies and the social rules by which they operate. Prof. Yochai Benkler (2007) refers to these frameworks from an economic perspective as *sharing economies* and introduces a term *commons-based peer production* to describe the structure in which information is produced. *Commons-based peer production* refers to the idea of a distributed set of individuals without the traditional hierarchy of a firm, i.e. peers, pooling their creative output into a commons that can then be used to support continued enrichment of that commons as well the goals of the participating individuals. In the case of free software, that commons is the source code that developers have made available to others under permissive (copyleft) licenses. The new economies that we are going to be discussing are also not market based, i.e. money is not the primary token of exchange. Instead "sharing" is the name of the game. We will look at some motivations present with these economies, taking free software as our example case, but first we will look at why these sharing economies are so salient in today's environment.

Political Affordances of Network Technology

As described by Benkler (2007), one of the major contributing factors to the salience of sharing economies is the technology that makes them viable. Relatively cheap and widely available tools for information production and the infrastructure that interconnects them, the Personal Computer and the Internet respectively, situate individuals much more centrally in the production process and further allow the coordination of these individuals' activities into groups and associations as large or larger than some of the biggest firms created by market based production systems. If we take the example of Wikipedia or the development of the GNU/Linux operating system, they work first because they are seated in systems (both technological and social) that empower individuals to use their talents and gifts *without* relying on a formal organization; for example editing or creating an encyclopedia article, or writing some code to fix a bug or add a new feature does not require the support of a company or organized group for those that have the knowledge to do so, and access to a (relatively inexpensive) PC. Secondly they are able to function due to the connectivity afforded by communications systems such as the Internet as well as socio-technical tools built to support and coordinate these activities across these geographically distributed networks, such as the wiki (both wiki-software and wiki-mores) or distributed version control, bug tracking software and bazaar style development philosophy (Raymond, 1997) associated with open source software. The affordances of these systems are political in the sense that they affect how people are able to orient themselves with respect to each other in a system of production.

The nature of the production of information goods also gives salience to the sharing model within this domain. In chapter two of this thesis we demonstrated with respect to art practice, the importance of being able to build on previous work in the creation of new culture. This is true of information in general; new information is constructed out of old. Technology that affords rapid and cheap transmission of information will encourage the creation of corresponding socio-economic models that set information free. The low [capital] cost of the method of production and low marginal cost of its reproduction, also allows us to sometimes completely escape the monetary reward aspect usually associated with getting something made. Other motivational factors are then able to come to the

forefront, such as "doing something for the love of it", or "the feeling of being in a community" or "doing something to make the world a better place".

Reward & Motivation in Free and Open Source Software

Given what we have said previously about the ability of sharing economies to sidestep monetary exchange, why do people do it? The simple answer is that money is not the only reason people 'work'. We shall try to unpack this statement a bit and look at a particular set of motivations for people to participate in free software. We can divide the rewards & motivations to participate in open source software into two broad categories, *individualistic* and *social* (in that they are borne out of interactions with others).Individualistic motivations identified in the literature include:

Learning. Where individuals will start or join an open source project in order to better learn how to do something. Linus Torvalds started writing the Linux kernel partly because he wanted to learn more about the architecture of the Intel 386 machine he had just obtained (Torvalds, 1991). While this motivation is often present when one is not working on open source projects, open source provides the extra opportunity of being able to look at other peoples code as well as get their feedback on your contributions. Hars and Ou (2002) identify that open source projects provide individuals opportunities to "select learning experiences that meet their needs and interests". Open source also provides chances for inexperienced programmers (e.g. students) to work on real projects[5].

Scratching one's own itch. This is a phrase coined by Eric Raymond, a writer and an experienced participant in open source, in his seminal text *The Cathedral and The Bazaar.* With respect to software it describes the act of creating or fixing something to solve a personal need, and is a common motivation in starting open source projects. (Raymond, 1997)

Personal Enjoyment. for many programmers, programming is not always 'work'. That is to say many programmers

5 A good example of this is the Google Sponsored "Summer of Code" which pairs students with mentors in open source projects to make improvements over the course of summer

find the creative and technically challenging nature of programming personally rewarding. (Torvalds & Diamond), 2002.

Politics. Part individual, part social, some free software participants do so out of the belief that it is highly important that free (as in speech — i.e. access to source code) software exist. These individuals are wary of the control over the means of using computers (i.e. Software) being completely under the control of commercial entities and corporations. (Lakhani & Wolf, 2005)

All the motivations above (except possibly the last one) are often true whether what one is writing is open source or not. Socially driven motivations on the other hand emerge out of the context of working within a group or community. Socially driven motivations include:

Consciousness of kind. This term describes a feeling of inclusion within a community and a felt connection between the members of that community as well as a collective sense of separation from non-members. (Bagozzi & Dhokalia, 2006, Lakhani & Wolf, 2005).

Sense of duty & Obligation. As developers and users (that support open source software use) become more central to the functioning of the community, they may begin to have a sense of duty or obligation towards the project, this obligation is not presented as a burden as it often represents a feeling of "indispensability" (von Krogh & von Hippel 2006, Bagozzi & Dhokalia 2006).

Recognition & Reputation. recognition of one's contributions are important personal motivations for participants in these communities as it creates a sense of appreciation for one's work which is an encouragement to produce more, it also provides a validation for the work one has put in to making a contribution. For example if one is recognized by having their patch accepted for inclusion into the main source, then that contribution is essentially blessed as being good enough for that community. When that validation is made public (in a contributor list for example) then that provides a public recognition that generates good feeling in the developer, the developer is also then able to leverage this recognition inside and outside of the community (getting other patches accepted, having more influence, or using that mention of contribution when

stating their qualifications in another context (Raymond, 1997, Lerner & Tirole, 2005). Surveys by Ghosh (2005) and Lakhani & Wolf (2005), confirm that reputation is an important motivator in open source.

Loosely Bound Cooperation

Our focus now shifts from the structure that we typically associate with the development of open source software to the mode of collaboration we are directly addressing in this thesis. We distinguish our use of the phrase loosely bound cooperation from the kind of collaboration one would see in Wikipedia or among Linux kernel developers (massive collaboration), by suggesting that in loosely bound cooperation individuals are pursuing their own *independent goals*, yet are able to *help each other along the way.*

While participants are not strongly bound to each other in open source development in the sense that participants can leave the project whenever they want (not without consequence, but they are not bound by contract). They are bound together in the sense that they are all working on the *same project*, indeed that is the reason that the community formed. In systems such as Wikipedia and GNU/Linux all participants are bound by (and participate in the construction of) a *shared vision* of what it is they are trying to create.

An example of loosely bound cooperation can be seen in the web service *delicious* ("delicious", n.d.). *Delicious* is a service that allows users to access their web bookmarks from any Internet accessible computer. Rather than store them locally on the users' machine, delicious allows you to store your bookmarks on their servers thus allowing you access to them from any computer that you can browse the Internet with. This goal is an individually motivated one — a user wants better access to their bookmarks. However in the context of a network of users, *delicious* is able to leverage this essentially selfish behavior (note that we do not intend to use the term selfish in a pejorative sense but merely to indicate that the behaviour is self-motivated) to provide added value for all users of the service. By allowing users to tag their bookmarks and by making them publicly searchable, *delicious* effectively provides a human filter on the larger Internet. By tracking simple metrics like who or how often

a bookmark has been saved to *delicious,* a popularity ranking is also created. 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 on the Internet. All this value is created 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 allows these users to cooperate with each other, and this is the model we want to support in our design. In considering loosely bound cooperation there is a wide continuum of relationships between the participants, from small groups actively collaborating or communicating with each other, to individuals on opposite peripheries of the domain that never make direct contact with each other.

Development Across Project Boundaries

When applied to software development loosely-bound cooperation can be seen in cooperative development *across* projects, and is something that can *already be observed* in the open source development ecosystem. One of the boons of open source development is that the source is not just available for you to change parts of a program you may not like, but also provides opportunity to reuse parts of that code in a completely different project. Richard Stallman writes about the programming community at MIT's AI Lab in the 70s,

> "We did not call our software "free software", because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, ***or cannibalize parts of it to make a new program.***" [Emphasis added] — Richard Stallman (Stallman, 1998)

This "cannibalization" is another example of loosely bound cooperation, the developers of the program were not building it to be cannibalized (but possibly to scratch their own itch), and the cannibal

does not join the formers community in the act of cannibalizing. The two are not bound to each other or to the same project yet they are able to cooperate (as one sided as the cooperation may seem at first) by virtue of the openness of the code. It is this use case that we seek to support in the design of *Share*. How do we design a system that supports this type of cooperation, and provides as much value as possible to all participants in the system?

Currently we see few to no tools that explicitly support this use case (Canonical's *Launchpad* project hosting service does provide mechanisms for reporting and tracking bugs that that affect multiple distinct projects); and methods to provide reward for these forms of cooperation are ad-hoc or non-existent. For example if I cannibalize a program, the original authors would never know unless I sent them an email or posted it somewhere in the credits of my program, and in the latter case the original authors may not even discover my program given that it is possibly in a completely different domain. This is not necessarily a problem, after all what I did with their code may not be interesting to them, but it is a potential lost opportunity to provide private or public recognition of their assistance with my work, as well as increased opportunity for them to see what uses their code gets applied to (and what changes may be made to it). Our proposal seeks to explore this space. While it is recognized that attribution is important among participants in the open source ecosystem, we seek to better encode this fact in the design of the programming tool itself.

Challenges Across Project Boundaries

When we contrast motivations for participating in this kind of loosely bound cooperation with that which is present within open source development projects a number of the social factors we identified in previous sections are missing. Firstly when individuals are not working on the same project, 'consciousness of kind' is reduced or non-existent as the participants do not form a cohesive group. Participants also carry little sense of obligation or duty with regards to other peoples' project. Additionally reputation cannot exist without a persistent identity within a community and a means of communication that allow markers of reputation to be transmitted and displayed. How can we deal with the even looser associations

between developers in different projects than those present in typical open source development? From the purely individualistic point of view there is less reason to *share* code that one has written with others doing possibly unrelated things. With this consideration we propose that systems like ours target existing communities of practice[6].

Wenger describes communities of practice as *"groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly"*. They are characterized by three things: a domain of interest, a community of people, and a shared practice (i.e. set of tools, experiences etc.) (Wenger, n.d.) He identifies several activities as being typical within communities of practice.

Problem Solving	"Can we work on this design and brainstorm some ideas; I'm stuck."
Requests for information	"Where can I find the code to connect to the server?"
Seeking experience	"Has anyone dealt with a customer in this situation?"
Reusing assets	"I have a proposal for a local area network I wrote for a client last year. I can send it to you and you can easily tweak it for this new client."
Coordination and synergy	"Can we combine our purchases of solvent to achieve bulk discounts?"
Discussing developments	"What do you think of the new CAD system? Does it really help?"
Documentation projects	"We have faced this problem five times now. Let us write it down once and for all."
Visits	"Can we come and see your after-school program? We need to establish one in our city."
Mapping knowledge and identifying gaps	"Who knows what, and what are we missing? What other groups should we connect with?"

(Wenger, n.d)

Communities of practice support the 'learning' motivation we identified in the previous chapter, however by virtue of being a community they are also able to provide reputation based motivation, and to a lesser degree than in cohesive projects the consciousness of kind from being a member of a community (albeit a broader one). Our proposed design seeks to reveal these patterns of sharing to better leverage the motivating effects of *reputation* in encouraging loosely bound cooperation.

6 That is not to say that we do not see *Share* as part of a toolkit that could be used in the development of a new community, but rather that in its current form we think that *share* complements the existing tools online communities use to interact.

Chapter 5. **Related Work**

Before describing our work we shall look at existing tools and projects that relate to how communities of programmers (and non programmers in the case of one of the projects) can share resources (typically code) with each other while being able to maintain various levels of independence. These tools also showcase mechanisms to recognize or otherwise reward participation and thus encourage participation in the community.

Processing, The Community; Forums and Other Existing Tools

As we have mentioned previously, our software was designed with a particular community of practice in mind, namely the *processing* community. Our inclusion of 'the community' in this related work section is to display some of the existing tools for loosely-bound cooperation available *in general* to communities whose interactions are mediated by the world wide web, using the *processing* community as an exemplar (though we will look at tools from other communities).

Personal web sites and blogs are a common method

for *processing* users to share the artworks and code experiments (processingblogs.org is a website that aggregates other blogs and websites that regularly post *processing* work), the software makes it fairly easy to publish one's work as it is a single click operation to generate the html code that embeds an applet version of a project built in *processing* (it is however up to the individual to find a method to host it), by default *processing* bundles the source files for the project along with the web page used to view it. Thus it is fairly common to find source code available alongside *processing* work when it is posted online (although many artists also remove the links to the source code, particularly for more mature works). Users also display their work (which is often visual) on photo and video sharing sites such as *flickr*[7] or *Vimeo*[8]. Recently sites like *OpenProcessing.org* have emerged that aim to provide hosting for the users' applets yet also emphasize a code sharing aspect; work uploaded to *OpenProcessing.org* must be licensed using a copyleft license with source code available for all to see. Through sites like these and the communication that occurs on them, members of the community are able to share knowledge, experience and resources with each other. However there are no mechanism to track activity across these resources and the actual programming tools, when we 'import' some code from a web site into our programming environment the link is lost, our project aims to better maintain these traces of reuse and connect the tools for collaboration to the those we use to write code.

Sharing information over the web is common in programming communities, where there will often be a number of widely read blogs in addition to mailing lists, forums and IRC channels. Within these forums participants are able to build a sense of community that encourages the flow of information. Donath (1998) has previously identified the importance of identity (and reputation) in motivating information exchange in user communities such as Usenet groups and identified features [and use practices] in the design of communication technologies that support the exposition of identity to the community, including seemingly simple things like email signatures or the domain from which an email is sent.

When we look at the design of forum software we see they often include some mechanism to explicitly display reputation. The image in fig. 13 below displays the panel displayed next to a user post

7 http://www.flickr.com/groups/processing/pool/
8 http://www.vimeo.com/groups/processing

on the Ubuntu forums, the coffee cup/bean icons are an indicator of this user's activity on the website, the forums also provide a mechanism for users to 'thank' other users for their assistance and for these 'thanks' to be displayed publicly, letting others know how helpful this user is. The StackOverflow ("Stack Overflow", 2009) community forums in addition to having a very explicit points based reputation system that allows users to perform different actions on the site, also has a system of 'badges' (fig 14) that users can obtain for particular behavior on the site.

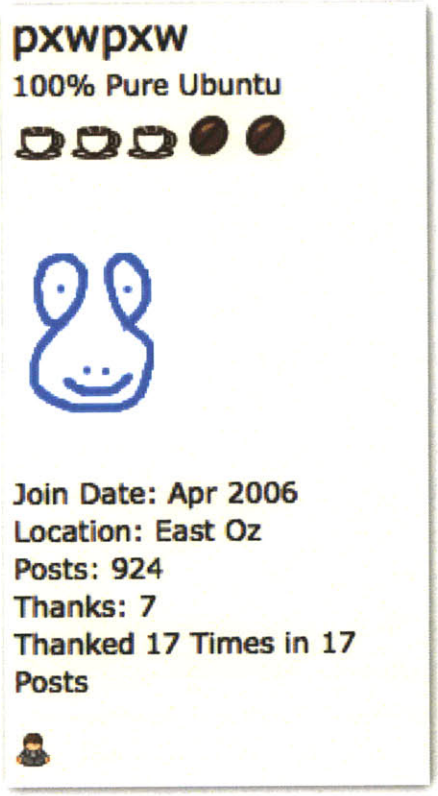

Figure 13. *Sidebar of a post on the Ubuntu Forums,* Retrieved on July 29 2009 from UbuntuForums.org, http://ubuntuforums.org/showthread.php?t=995704

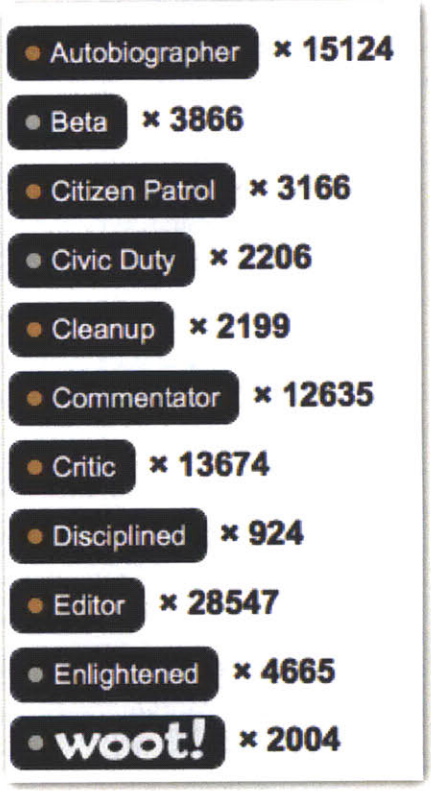

Figure 14. *Sample of some of the badges available to Stack Overflow users, the numbers beside the badge indicate how many users have obtained that badge.* Retrieved on July 29 2009 from StackOverflow.com, http://stackoverflow.com/badges

These are both examples of mechanisms that use reputation to motivate people to participate in these forms of loosely bound cooperation and perform the tasks desirable to the systems creators. We hope that our visualization of attribution will perform a similar role in encouraging code sharing. While the *processing* forums are much less overt about displaying user history (they are not displayed on every post but only on a user profile page), it is important to note that these are design decisions that shape the way the community behaves. Our work, like the *processing* forums, shies away from the explicitly competitive reward and recognition mechanisms.

Scratch

Scratch (Maloney et al, 2004) is a programming language and community geared towards children, that provides a lot of encouragement to share one's work. Youth are able to create Scratch projects and easily upload them to the Scratch website where they are put on display for other members of the community. Other youth can then download these projects *and their source code* and modify them in the creation of new work. When they upload these new projects they are marked as remixes, thus providing attribution to the original source. The integration between the tool and online community are right in line with what we are trying to achieve in our work. Monroy-Hernández (2007) identified the usefulness of *creative appropriation* for learning in communal environments and describes the ways in which the design of the scratch website encourage participation in communal exchange. The scratch website encourages uploading work to the site by highlighting works across various metrics, including "Top Remixed", "Top Loved" (a popularity metric), and "Top Downloaded" (fig. 15). By placing these projects prominently on the home page it provides great reward for projects that successfully engage with other members of the community.

Figure 15. *Popular projects highlighted on the Scratch front page.* Retrieved on July 30 2009 from scratch.mit.edu, http://scratch.mit.edu/

Our work seeks to extend the practice of creative appropriation by increasing the granularity of artifacts that can be shared. The main shared artifact in Scratch is the whole project, and while a user can remix an entire project, it is currently more challenging to make use of a component of one project in another one. This is done by exporting "sprites" from one project and importing them into another, however during this process the link between the two projects is not captured by the system. To continue the use of the musical metaphor, if Scratch enables individuals to track remixes our work seeks to track sampling.

OPENSTUDIO

Described as a "an experiment in Creativity, Collaboration and Capitalism", *OPENSTUDIO* ("OpenStudio", n.d.) couples a simple drawing tool "with an economy of artists, curators, collectors, dealers and viewers". Users of OPENSTUDIO create artwork using a simple drawing tool that they then sell or give to others. Users were also allowed to modify (remix) work they had bought before selling it on. Coupled with a *process capture* tool created by Ding (2006) that recorded the various steps in creating a drawing, they were able to display the history of an art piece as well as who had done what in the creation of a derivative work. Not only does this support learning from watching other peoples' drawing process, it provides publicly visible attribution when derivative works are created (they even report success at detecting forgeries and blatant copies). However the model implemented in OPENSTUDIO made a rival good out of the digital work produced, when a work was sold, ownership and control were completely transferred. Also OPENSTUDIO operated primarily in a market economy (including a virtual currency, the "burak") as opposed to the non-market sharing economies in which we are interested, thus different motivational factors are present. It is however an interesting study in how a system like *Share* (which has similarities with OPENSTUDIO — minus the virtual economy), might better interact with the commercial reality of some of the participants that may use it.

GitHub

GitHub ("GitHub", 2009) 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 'forks' to projects you put up when the 'forker' is a user of the github service as well (a fork in github terminology is a branch of a project created and controlled by another user). One important difference with what we are trying to achieve is around the issue of working across projects. The concept of a 'fork' does not really exist when integrating pieces of diverse projects with something you are working on. Similar to our comparison with Scratch, we seek to examine a design that works at a finer level of granularity.

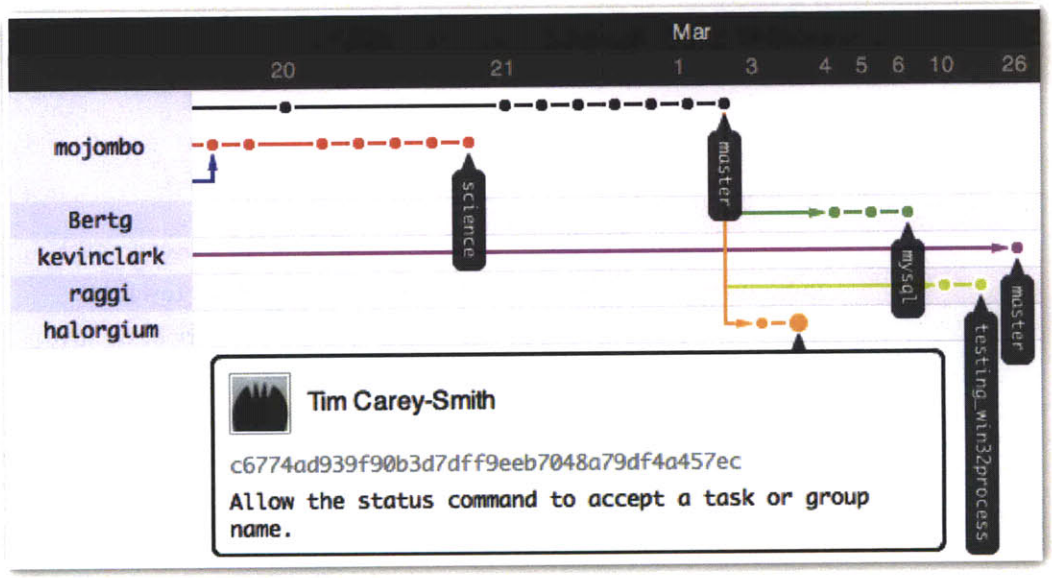

Figure 16. *View showing forked versions of a project on github and the commit activity on them.* Retrieved on July 30 2009 from http://github.com/blog/39-say-hello-to-the-network-graph-visualizer

Chapter 6. **Design & Implementation**

As stated in the introduction, this thesis proposes a novel programming environment, titled *Share,* that is geared towards encouraging loosely bound cooperation in a community of creative hackers. We aim to do this by leveraging the recognition/reputation related motivational factors present within communities of practice as well as providing pragmatic (i.e. related to the actual practice of programming) value for sharing code. Our programming environment has a number of features towards that end: it automatically shares all the code written in it with other members of the community and also tracks movement of that code (via copy-paste operations) to a fine level of granularity, such that the system can identify the source of any given character within the repository. Users are also able to explicitly make links to other code artifacts.

The relationships thusly built between code artifacts are visualized to users of the tool, providing a form of automatic public attribution as well as a more socially oriented way to browse through a code repository. *Share* also provides a simple form of persistent conversation around each code artifact.

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. Because one is working in public view, it is important the space not privilege finished work over work in progress. Thus in *share* code is continuously uploaded as it is written (as opposed to when it is deemed finished by its author).

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). As users of the tool are pursuing their own independent goals, they should not feel disrupted by the environment or other users, at the same time however we want to provide a smooth continuum for increased engagement with the community that the tool connects you to. This design goal led to a focus on interaction in *share* being asynchronous and a decision that one should be able to program in *share* whether one is online or offline, with seamless transition between the two states.

Non-competitiveness. Given our use of attribution and reputation 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 workspace. We desired to create an attribution system that would not encourage overly competitive behavior. To demonstrate the variety of approaches one can take in designing such a system, fig. 17 produced by the Yahoo!® corporation displays a library of 'patterns' commonly used in creating a reputation system ("Yahoo! Design Pattern Library- Reputation", n.d.) and even organizes them along a competitive spectrum ("The Competitive Spectrum Pattern - Yahoo! Design Pattern Library", n.d.). We mainly have tried to focus on subtle displays of reputation.

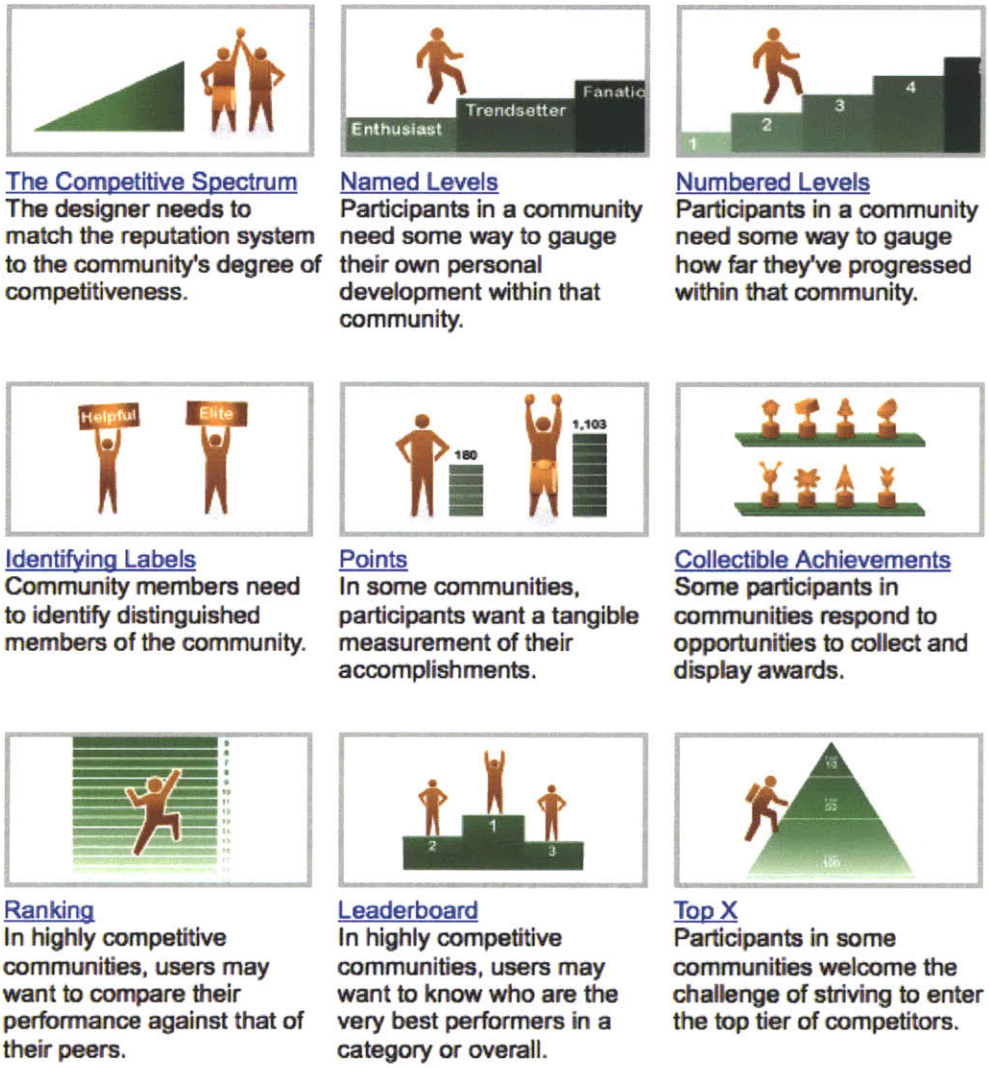

Figure 17. *Reputation systems from Yahoo Design Patterns Library.* Retrieved on July 27 2009 from Yahoo.com, http://developer.yahoo.com/ypatterns/parent.php?pattern=reputation

For example we would be avoiding patterns like *ranking*, *leaderboard*, or *points* while others like *collectible achievements* or *identifying labels* may be more suitable.

Cross-platform support. For practical reasons relating to being able to recruit users we wanted *Share* to run on all three major platforms supported by *processing*. Windows, Mac OS X and Linux.

Thick Client / Thin Server. We decided that most of the

work would be done in the client, with the server playing as minimal a role as possible. The main reason for this was to support easy deployment of *share* servers and to reduce the hardware requirements related to scaling the server component to supporting larger numbers of users. Our current experience with carrying out experiments 'in the wild' make us sensitive to the fact that when one proposes a tool for community use and invites use of the tool there may be an opportunity for continued use of the 'experimental' tool by the community once the study is complete. We had always planned on releasing *share* as an open source project and wanted to minimize the requirements of setting up a server and thus allowing for continued use. The following subsections describe the design of the system we built as well as some implementation notes.

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 to and from which documents and data files associated with projects are pushed and pulled.

Share Server

Share's server component consists of two main parts: a CouchDB database and a small Sinatra based ruby web application that controls authentication when pushing documents to the server. CouchDB is an HTTP accessible, schemaless, document-oriented database ("Apache CouchDB", 2008); it is essentially a key value store with strings for keys and JSON[9] documents as values. A number of CouchDB features led us to select it for our server-side persistence solution, including a few that would allow for relatively easy scaling to larger numbers of users without complex infrastructure.

Firstly, it is designed to support highly concurrent loads. Written in erlang, a programming language at developed at Ericsson to build highly concurrent applications, it's multi-version concurrency control architecture allows writes to be performed to the database without blocking reads, yet presenting a consistent state to clients. Our previous experience building client-server web applications informs us that database access is often the bottleneck to achieving good communication throughput between clients and the server, CouchDB is a database designed with these issues in mind.

Secondly CouchDB is accessible over HTTP (HyperText Transfer Protocol), this allows the clients to talk directly to the database with a simple protocol using http libraries commonly available in any modern programming language. By allowing clients to talk directly to the database we do not have to maintain an intermediate layer to handle these requests supporting out thick-client thin-server design goal.

Thirdly, CouchDB includes the concept of views, these are map-reduce[10] jobs written in JavaScript that allow us to compute values based on the data stored in the database. This processing allows us to eliminate duplication of data when we want to be able to query the data in multiple ways (for example finding out which projects a user has commented on, as well as which users have commented on a particular project would require a bi-directional map (or storing two maps) but with CouchDB we can compute one from the other on the

9 JSON is JavaScript Object Notation a lightweight data-interchange format, www.json.org

10 "MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key." (Dean, Ghemawat, 2004)

fly). These views make the basic key value store a bit more versatile.

Finally CouchDB supports easy replication of data between instances of CouchDB, this would allow individuals or groups to set up their own *share* servers and periodically synchronize with other servers (essentially forming a peer network of servers), while we didn't need this for our initial evaluation it is nice to have and allows the underlying data storage architecture to be spread throughout the community if so desired.

Although CouchDB is still in alpha (we were using version 0.9) it is fairly stable and quite functional, however it does not currently support any means of authentication and because we wanted to allow clients direct http access to read from the database this meant that we would be giving clients uncontrolled write permissions to the database as well. This is obviously undesirable as it allows anyone with an http client (including common command line tools like curl) to edit the database. Additionally because we had always planned on open sourcing the code, it would allow individuals to modify the client such that they could make modifications to other people's documents and have those pushed out to the server. To work around this we setup the *nginx*[11] web server to work as a reverse-proxy in between the clients and the database, passing the http requests on to the database, however we configured it to only allows GET requests coming from sources other than localhost to pass through to the database (effectively limiting unfettered access to the database to just reads). *Share* clients thus submit POST requests (adding or updating documents), to a ruby application running on the same machine as the database that first does password authentication for the request then forwards it to the CouchDB instance, passing the result back to the client. While we lose some of the concurrency benefits provided by erlang when dealing writes, being such a simple operation it is quite easy to scale, and we can still have clients reading from the database while we are writing to it (it is actually fast enough for our current needs that we currently run the website for *share* in the same ruby process). Before downloading *share* users make an account on the website to create a username/password combination.

11 Nginx (pronounced engine-x) is a small fast web server similar in function to Apache. http://nginx.net/

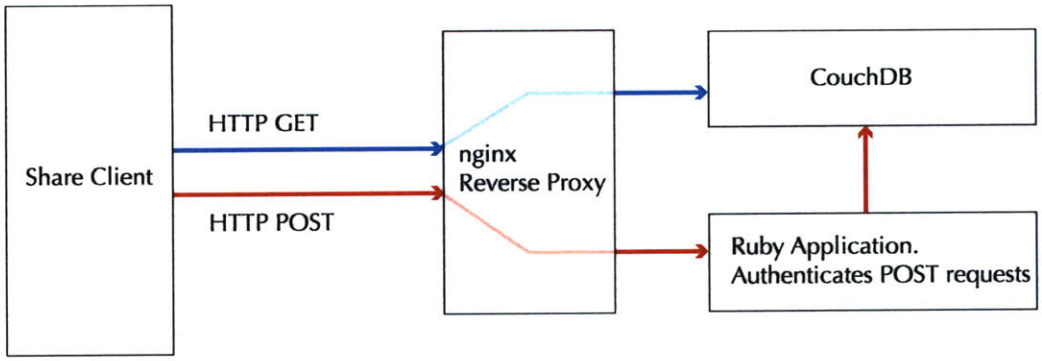

Figure 18. *"Share" client server architecture.*

We store a several different types of 'documents' in the database. Namely:

1. A document for each code file in *share*
2. A document for each data file included in a project
3. A per-project document recording comments on that project
4. A per-project document of bookmarks on that project

Documents are inserted into in the database with a Universally Unique Identifier (UUID) as their key, these UUIDs are generated by the clients, this allows documents to be created and ID's to be generated without being connected to the server yet avoid conflicts when syncs are done with the server, allowing the client to work completely disconnected from the server, supporting our non-disruptiveness design goal (you could program in *share* just as you would in *processing*, on the bus or at the beach). Simple mappings are used to create unique keys for the documents associated with projects or files. For example the document recording comments for a project with ID 'foo' will be given the key 'foo_comments', allowing easy retrieval.

Share Client

The *share* client is the program that the user primarily interacts with; the following subsections will describe the functionality of various parts of the software. The client is written in the Ruby programming language, using its Java implementation, JRuby. Using JRuby allows us write code in Ruby yet easily leverage mature java libraries such as Swing (a GUI toolkit) or Lucene (a full-text search engine). A high level overview of the subsystems within the client is displayed in the table below.

Non User Facing Subsystems	
Synchronization	Pushes changed documents to the server while pulling newly updated documents from server.
Toolchain	Responsible for taking project source and turning it into an executable. We currently include a *processing* toolchain.
Graph Model	Internal model of the users, projects and files. We also dynamically build a model of relationships between projects.
Search Engine	Provides full text search to users and also used as a query engine internally in building the graph model.
Persistence	Serializes and deserializes the code and metadata stored by the code editor to and from disk.
User Facing Subsystems	
Code Editor	Allows users to write and edit code, keeps track of provenance of code. Provides UI for commenting and bookmarking.
File Browser	List based UI for browsing through users and their projects.
Search Window	Full text search of all code generated by users of share.
Visualization	Interactive visualization of the relationships between files.

Table 1. *"Share" client subsystems.*

Our discussion of the client in this chapter will focus on the users' experience of using the software and less on how the backend works.

File Browser

The file browser is the first thing a user sees after logging in to *share* and is their entry way to opening other types of windows including the code editor or search panes.

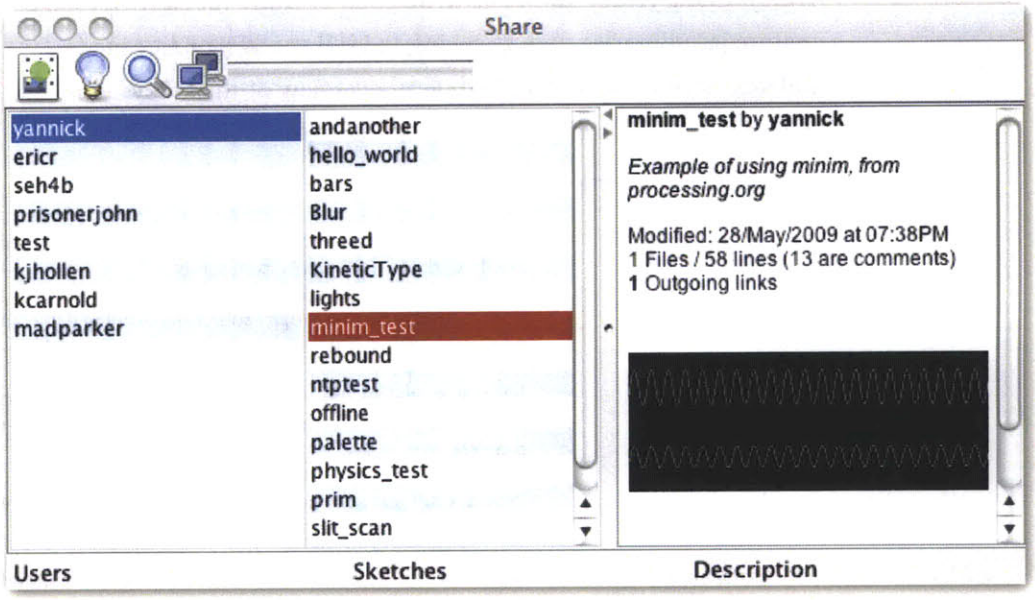

Figure 19. *"Share" File Browser.*

The file browser displays two lists to allow users to look through other users' projects (sketches), and 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 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. While we do not provide a global list of the most 'popular' projects, when a user is browsing they can get a sense of a projects' relative popularity. This is one of our forms of 'subtle' reputation.

Double clicking on a project name will open it in the editor. The file browser's toolbar also has buttons for creating a new project, visualizing a project's connections in the network browser, searching for code within the code base, and performing synchronization with the server. This last button does not actually need to be used as

synchronization is automatically performed every five minutes, but is there in case the user wants to force synchronization (for example just before exiting).

Code Search

Share provides full text search of the entire code base using the Lucene text search library ("Apache Lucene", 2006). The code search window allows users to search for code using standard boolean search operators; wildcards can also be used. Search results can be previewed in this window and projects directly opened from it.

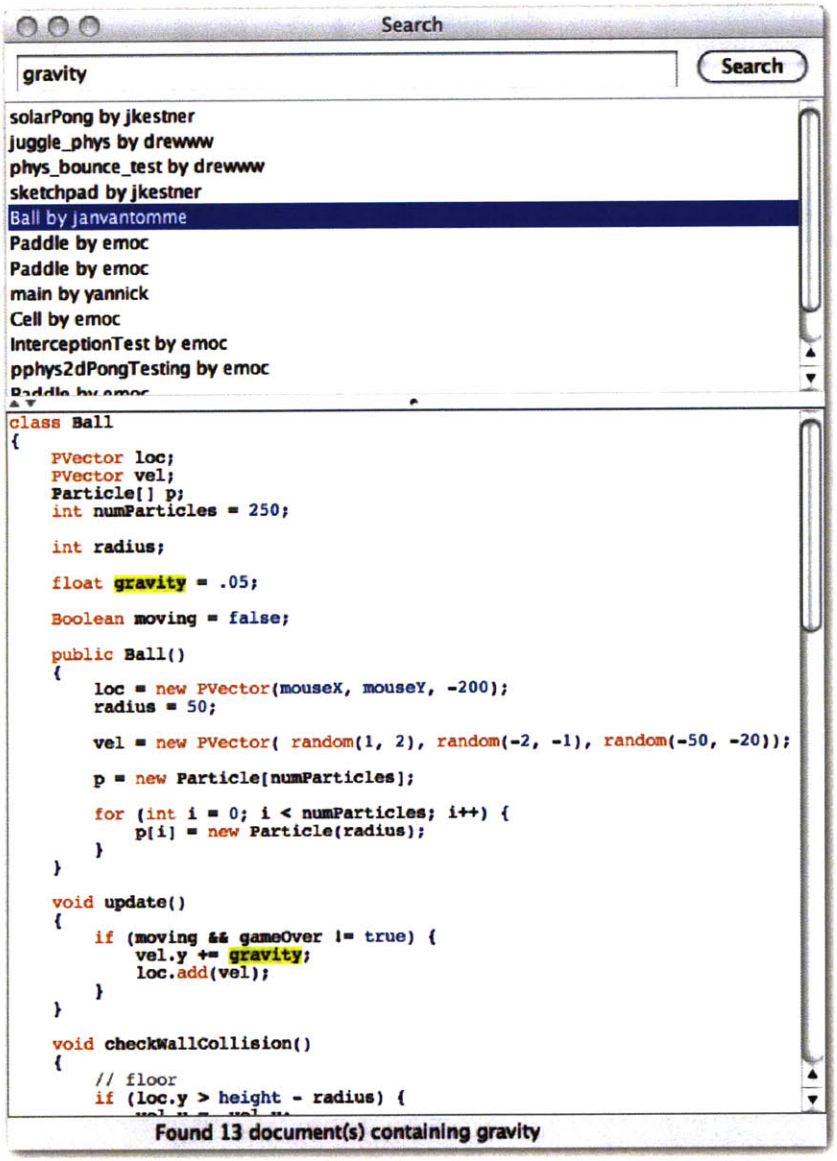

Figure 20. *"Share" Search Pane, search term is highlighted in results.*

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. As far as code editing features it is a fairly simple code editor, our goals in this regard was to reach parity with the editing features of the *processing* IDE, which also provides a fairly simple code editor. It provides syntax highlighting and smart indenting, though, unlike the *processing* IDE it does not provide brace matching or an automatic formatting tool. The *processing* IDE also provides tools like a color picker and a font creation tool that we do not. Like *processing*, the editor supports tabs (multiple files in a project) and has a 'play' button to easily compile and launch an application.

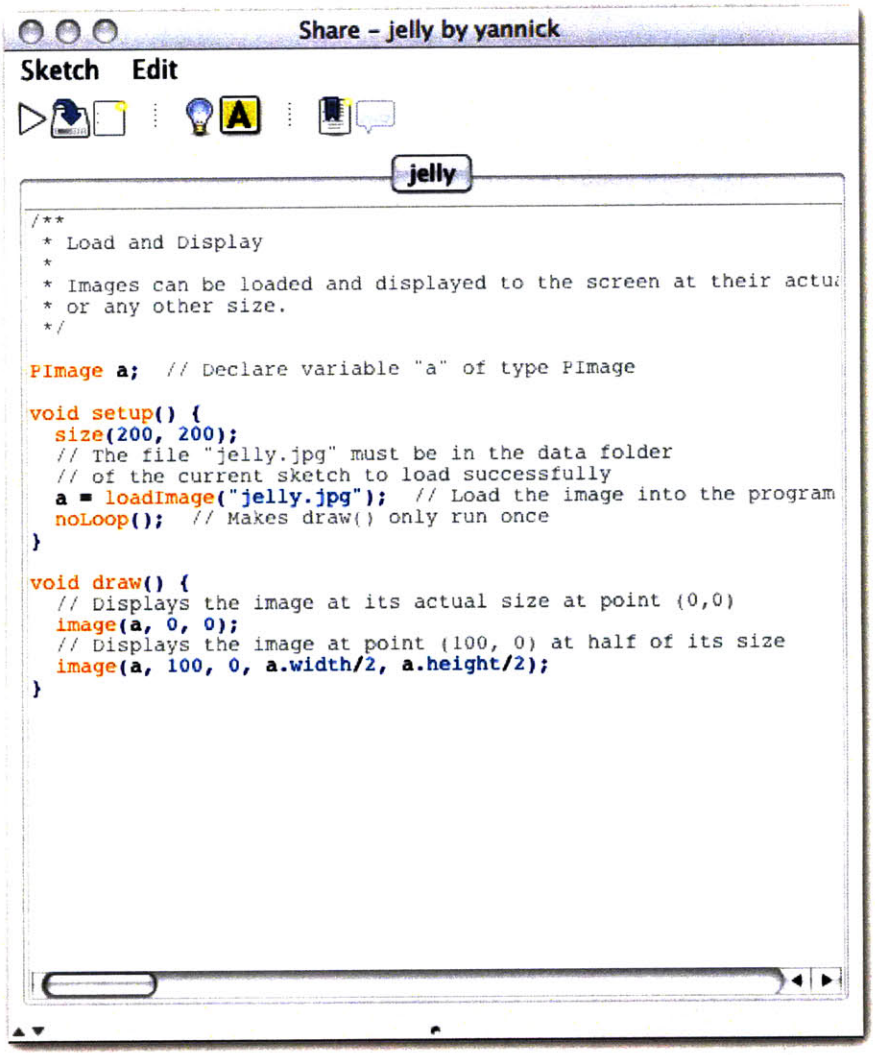

Figure 21. *"Share" Code Editor.*

The editor is also able to record attributes on each character of the 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. As we mentioned previously all documents and projects are given universally unique identifiers (UUID's), these UUID's are also used to name the files when written to disk, this also helps prevent issues with the different file names permissible across the operating systems we support. 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. Metadata such as the human readable name of the document of project are also stored in the files when they are saved. The text is written to an XML based format that lets us persist and restore the attribution information. An example of the representation underlying the code in the editor is given below.

```
<document id="N7b8270d_9f55_4534_
a8cb_126f6f74e922" sketch_id="N0bcf045_cdfe_4810_
b934_a60ea086f8e3" name="main" synched="true"
primary="true" modified="Sun Jun 14 00:17:11 UTC
2009" user_name="yannick" language="processing"
sketch_name="prince_two">
<content>
  <npe_code source="N7b8270d_9f55_4534_
a8cb_126f6f74e922" >
  //Import all Phys2D libraries
  import pphys2d.bodies.*;
  import pphys2d.joints.*;
  //Create a PPhys2D world
  PPWorld world = new PPWorld();
  </npe_code>
  <npe_code source="Ndf1b1f3_f470_4928_abf0_
c5d9aa8bd287" inserted_at="Thu Jun 11 14:37:20
-0400 2009" >ArrayList </npe_code>
  <npe_code source="N7b8270d_9f55_4534_
a8cb_126f6f74e922" >planets;</npe_code>
</content>
</Document>
```

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 the screenshot below.

Figure 22. *"Share" Code Editor in source highlighting mode.*

In this screenshot 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 (again this is a local assignment) 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 using a button on the toolbar.

The XML representation also allows us to use Lucene's full text search to determine the relationships between files as we load them from disk. For example to find all the documents that have

outgoing links from a particular document foo (i.e. documents that have borrowed code from 'foo') we can ask Lucene to search for files with `<npe_code source="ID of foo">` in the underlying representation.

Permissions

By design users are not allowed to modify other users' code, however to support quick experimentation with code of interest, the editor will allow you to make non-persistent changes to projects that are not yours, and to run the code with those changes. The editor lets users know when their edits are non persistent.

Explicit References

Another feature of *share* is the ability to make explicit references to other projects, this is because we believe that there are things that one may want to attribute others for that aren't in the code per se, such as ideas, inspiration and maybe even techniques that individuals may get as they look at other peoples code. This is done using a special syntax directly in the file, the syntax consists of using the @saw keyword and then giving a username/project pair. The following code "@saw yannick.rebound" will create a link between the document it is written in and the "rebound" project created by user "yannick". One can optionally append a filename to link to a particular file within the project such as "yannick.rebound. physicsEngine", we chose the word @saw so that it could be easily used in a sentence that would describe the reason for the reference, for example "I @saw yannick.rebound and figured that the ball could get faster each time it hits the wall". We experimented at first with a variety of keywords that could be used to indicate different kind of links but decided that it would be best to first release the tool and see if a lexicon emerged among users describing the kinds of references people would like to make, thus @saw also had the advantage of being a highly flexible keyword.

Comments

Share allows for discussion of code to stay near the code. Each sketch has its own comment thread accessible from the code editor. Inspired by Wikipedia's 'talk' pages, the idea of having persistent conversation about code happen in the context code itself is one we are very interested in but are just scratching the surface of in our current implementation. You currently need to be online to post or view comments.

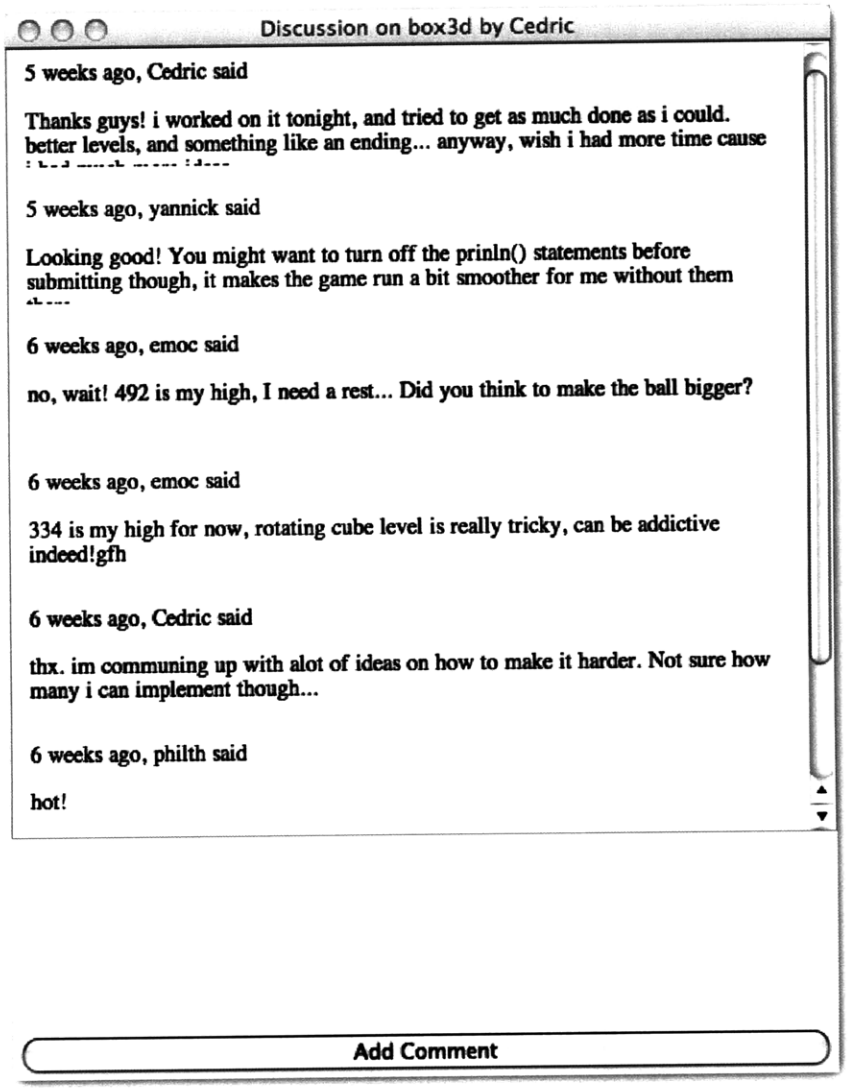

Figure 23. *Comment pane for "box3d" sketch.*

Bookmarks

Share also allows individuals to bookmark sketches that they find interesting or want to keep a reference to (it is also another 'subtle' reputation marker - while there are no global lists of what has been bookmarked the most, you can see how many times something has been bookmarked). A small dialog is used to allow users to bookmark projects and optionally add a small annotation to the bookmark. Bookmarks and their annotations are publicly visible to anyone using *share*.

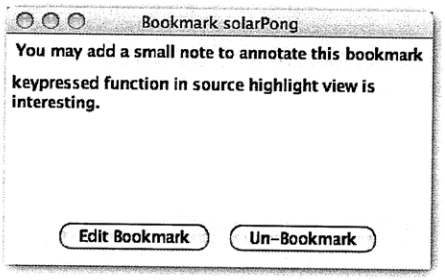

Figure 24. *Bookmark pane for "solarPong" sketch.*

Individuals can browse their bookmarks and the bookmarks of others using the search pane. The following query is used to see one's bookmarks "@bookmarks"

Search

@bookmarks

Search

solarPong by jkestner - keypressed function in source highlight view is interesting.
TwitterTest by toxi
another by yannick - rebmark

```
    grav.y = constrain(grav.y,-90,90);

    world.setGravity(grav.x,grav.y);

   }

}

void mousePressed() {
  throwAnchorX = mouseX;
  throwAnchorY = mouseY;
}
void mouseReleased() {
  addComet(mouseX,
mouseY,(mouseX-throwAnchorX)*10000,(mouseY-throwAnchorY)*10000);
}


void addComet(int x, int y, int forceX, int forceY) {
  println(forceX+" "+forceY);
  comets = (Comet[]) append(comets, new Comet(x,y,forceX,forceY));

}

void addCelestialBody(float diameter, int mass, int distance, float speed,
Color c) {
  bodies = (CelestialBody[]) append(bodies, new
CelestialBody(diameter,mass,distance,speed,c));
}
void addCelestialBody(float diameter, int mass, int distance, float speed,
PImage p) {
  bodies = (CelestialBody[]) append(bodies, new
CelestialBody(diameter,mass,distance,speed,p));
}

void drawGravityIndicator() {
  pushMatrix();
```

Found 3 sketches in yannick's bookmark list

Figure 25. *Search pane being used to display bookmarks for the logged in user.*

You can also view someone else's bookmarks using this query by appending their username, thus "@bookmarks yannick" will display the bookmark list for the user "yannick". You currently need to be online to post or view bookmarks.

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. It works by maintaining a list of UUID's of the documents and their corresponding revision numbers. Revision numbers are given to documents by the CouchDB database and are updated automatically whenever a document is updated on the server. Thus if the local client either does not have a revision number for a document, or it has a number that is different from what is reported by the database as the latest revision number, the document is pushed to the server or a new version is downloaded respectively. At every sync the client queries the database for a list of key value pairs containing all the ID's of all the documents stored in the database and their revision numbers, thus the client can also find out about documents that it has not seen before. When a document is downloaded its revision number is updated and written to disk. This mechanism allows us to only download documents that have changed. A similar mechanism is used to synchronize arbitrary data files that are used by projects.

The Network Browser

The network browser is an interactive visualization of the relationships between the projects in *share* and their owners. It acts as another 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 tree is built using a breadth first search, the depth of which is limited to six (greater depths would not be visible in the current set of visualizations). The process of creating the tree from the more general graph potentially eliminates some of the links with the graph, however we feel that this representation more clearly shows the elements that are most closely related to the selected node and also provides a clear means to browse the relationships in the graph by successively moving outward without us having to worry about trying to fit the whole graph on screen. The code editor and file browser allow users to visualize the connections for any user or project. The visualizations were implemented using the *processing* library (yes that is the same *processing* as the language that *share* currently supports, one can also use the *processing* api as a standard Java library, and thanks to JRuby we get to use it while still writing ruby code!)

There are two visualizations provided by the network browser, the first is a radial tree view with the selected entity placed in the center, the algorithm used is a partial implementation of Yee et al's (2001) layout algorithm for radial graphs that we ported from Jeffrey Heer's *"prefuse"* visualization library (Heer, Card & Landay, 2005) (our implementation does not animate the motion of the nodes using polar coordinates). In this visualization successive rings display entities directly related to an entity on inner ring. Relationships shown include those where code has moved between projects, those where '@saw' references have been made and creator/project relationships. In the case of code movement links, the arrowhead points 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; thus if a project gets a lot of its code (relative not absolute) from another, the line will be fairly thick. We had started with all arrows being the same thickness as we were wary of the meaningfulness of lines of code as representing the 'magnitude' of the contribution (sometimes one does find small but critical amounts

of code), however on showing this to early testers the feedback we got was that people did want some notion of the 'importance' of the link and the amount of code involved. @Saw links are drawn with dashed lines, and creator/projects are drawn with faint dotted lines, we also use color to relate project icons to the icons of their creator. The two are rendered in the same color (the logged in user is always rendered using black) and this is the same color used in the source highlighting view in the code editor. The icons for projects also display the total number of incoming and outgoing links that they have, remember that since we are looking at a tree built from the perspective of the selected node, there are cases when a distant nodes' linked nodes are already in the graph and as a tree cannot have cycles, this second link cannot be drawn.

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 to the edges if necessary. Allowing the user to progressively move closer and closer to the edges of the original tree. Originally a user could double click on the icon for a project to open it in the code editor, however two days before the start of our public release of the software we discovered a threading issue between the windowing toolkit (Swing) and the library driving the animation (*Processing*) that caused the application to freeze when opening large files from directly from the network browser. We were unable to solve the problem and had to disable this feature for the release. This meant that if one was interested in seeing the code for the project one had to open it through the file browser.

This visualization is also aimed at supporting discovery of previously unknown resources, this is why we do the breadth first search to a depth of six and expand user nodes as we find them, displaying other work created by users in the selected nodes graph. As you can see in the screenshot below there are a lot of nodes that are only indirectly related to the selected node, while we did not implement any filtering to control the number of nodes shown in the visualization this would become more and more 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.

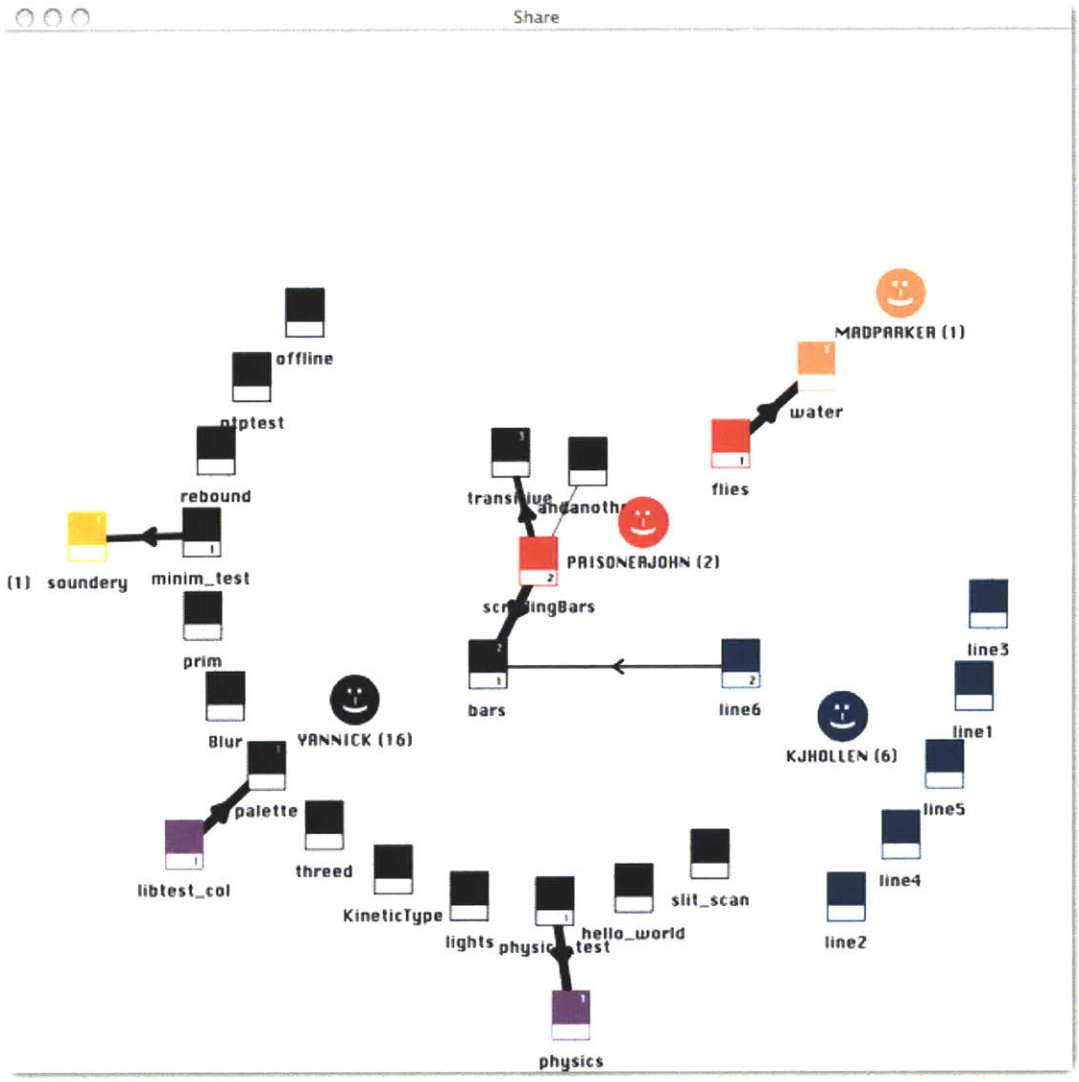

Figure 26. *Network browser visualizing a netowrk of sketches.*

Users can toggle a second visualization that is much simpler than the radial browser, which simply answers the question "what projects are contributing to and borrowing from a project". It thus shows elements that are only one step away from the selected node. Like the radial browser when a new node is selected it smoothly animates to the center and any new related nodes fade in and animate into place.

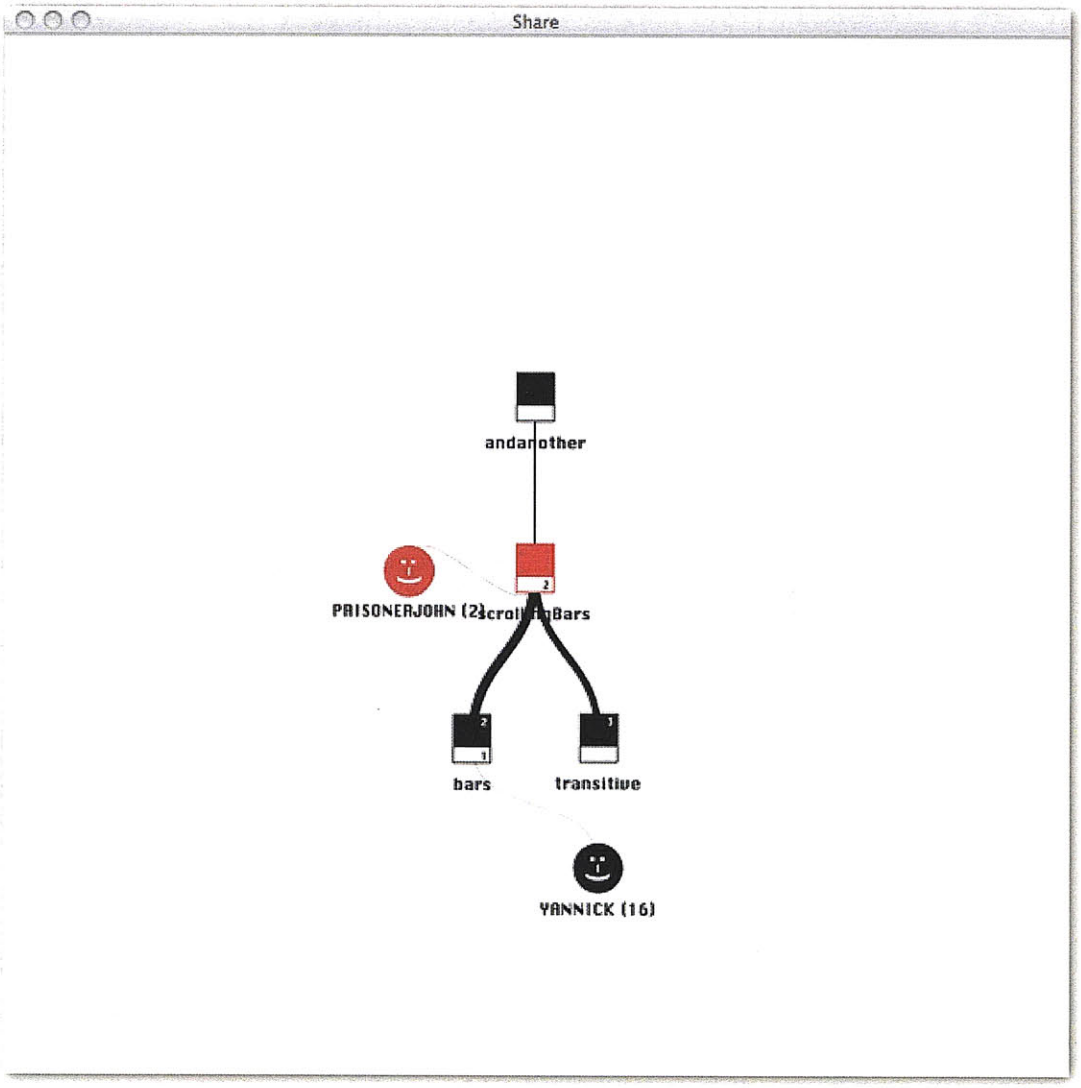

Figure 27. *Network browser in secondary mode, displaying only incoming and ourgoing links to focused sketch.*

Runtime

Share ships with the *processing* compiler and runtime, however it is architected in such a manner that it can easily support the runtimes and toolchains of other programming languages. When a project is launched in *share*. We do a small amount of preprocessing to first strip the metadata that we maintain for each document, we then create a folder with all of the source and data files as the compiler would expect (essentially like it had been created in an ordinary text editor). We then pass this older to the *processing* compiler and runtime, which takes care of the rest.

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 key that will take a screenshot of their project and name associate it with the project so that it can be seen in the file browser. When a running program is terminated, *share* checks for any new screenshots and imports them into its folder structure for later synchronization with the server.

Security Concerns

One of the issues in a system like *share* is that when a user run another user's sketch they are essentially running code from some "random person on the internet", this is definitely a source of concern and while one mitigating factor is that one has the source code to any program one would run right in front of them, it doesn't adequately protect users who may not be experienced enough to detect malicious code. It would also be time consuming and error prone even for advanced users. One advantage of *processing* being a language built a top the Java Virtual Machine (JVM), is that Java has strong mechanisms to restrict the actions of programs running in it. All programs run from *share* are run in a sandbox managed by the JVM, the security policy we set when launching sketches only allows them to read and write to files in the directory from which they were launched, network operations are also blocked as is the ability to launch other programs. This lets us provide a great amount of safety to users of *share* however it does prevent some non-malicious programs from running, for example those that want to use the computers webcam or (non-maliciously) connect to some online

resource. We thus provide an option to run programs outside of the sandbox, though users are warned to inspect or otherwise ensure that programs they run in this 'trusted' mode are indeed not malicious.

Chapter 7. The Share Experiment: Results & Discussion

In order to evaluate our design we made our system available to a small number of users for a two-week period. This chapter describes the framework in which we carried out our evaluation as well as responses from users to a survey provided once the two-week period was complete. Our goals in deploying the application was to gauge user response to our design choices as well as answer the research questions set forth in chapter one of this thesis. It is on the users' responses to our survey that this chapter most strongly focuses; our survey questions focus on users' impressions of the attribution features provided by *share*, the usefulness of the ability to track re-use of one's own code and the disruptiveness of the community oriented features provided by *share*. We also had opportunity to ask participants about their previous code sharing experience and the motivations and obstacles in doing so. Overall we are able to report that users valued the feature set provided by *share* and that it did in fact reduce their barriers to sharing code.

Experiment Design

While systems such as the one proposed in this thesis are best evaluated in the context of long term use within a community, time constraints and a desire to get initial feedback on our prototype led us to propose hosting a themed competition to evaluate our design. The purpose of the competition structure and theme was to scaffold the creation a small scale community of practice which provides the loose associations and shared interests we would expect to see in larger communities of practice but does so in a manner that makes the short timeframe analysis more practical.

Participants in our competition, dubbed *The Share Experiment,* were asked to create works 'Inspired by Pong'[12], this was the only constraint given with respect to creative work. Participants were also told that they could interpret the theme quite broadly and their creations did not necessarily even have to be games. The participants were given two weeks 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 the interaction we would expect in longer-term deployments *share*. Of the 16 participants that ultimately participated in the competition 11 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). Prizes were offered as incentive for participation with four 'grand' prizes on offer including two Apple iPod's and two Arduino[13] Kits. While a competition was used to recruit and encourage participants to actively use our software, this project is much more about cooperation than competition so we also stated that a $25 gift certificate would be awarded to any user whose code was used by a winning submission. This meant that a person borrowing your code simply increases your chances of winning something, and was in very much in keeping with the spirit of *share*.

Participation levels varied among users with some users being very active and others who only spent a few days using *share* either at the beginning of the experiment or towards the end. 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 survey used is included

12 Pong is an early arcade game. http://en.wikipedia.org/wiki/Pong
13 Arduino is an electronics prototyping platform http://www.arduino.cc/

in Appendix A of this thesis. The investigator also interacted with the participants throughout the course of the event, the results discussed in this chapter come from three main sources, the metadata on code sharing collected by the software (data from all sixteen participants), the participants responses to the questionnaire (from the eleven respondents) and the investigator's correspondence with participants during and after the competition. Our discussion will try and incorporate the responses of participants in their own words as much as possible.

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. A call was made to solicit 30 participants by making a post on the *processing.org* forums as well as submitting our call to a widely read art and technology blog *Networked Performance* (Green, Thorington, Riel, 2004) from which we received over 70 applications. After sending out 34 invitations we received 28 confirmations, however only 16 of the confirmed participants downloaded and used the software.

Our criteria for selecting individuals to participate in the study were primarily along two axes, experience with programming and the individuals' relationship to *processing* in their work/study life. Specifically we asked applicants to rate themselves as beginner, intermediate, or expert. We also asked applicants what occupation they currently held and whether they used *processing* in their work/study life. Our aim was to get a broad range of participants along both axes as we felt that the value proposition for *share* would be different for beginners vs. experts and we were also interested in the different motivations and barriers to sharing that would be present between those that use *processing* primarily casually vs. those who use it in their professional life. Participant experience breaks down as follows.

Beginner	3
Intermediate	10
Advanced	3

With regard to professional use we coded responses into 3 categories, academic (teaching or studying), commercial (including independent artists but not including teaching) and casual use. While participants may use *processing* in more than one category we assigned each participant to one of them, picking a dominant category according to the following rules; membership in the commercial category trumped membership in the academic category, which in turn trumped membership in the casual category. We did this because we felt that commercial factors would likely be the strongest ones with respect to code sharing barriers. Participant breakdown along this axis is as follows.

Academic	6
Commercial	6
Casual	4

However given the low number of responses we received to the survey we do not feel that we are able to make strong conclusions with regard to the effect of professional relationship to *processing* on response to the feature set provided by *share* or the difference in value proposition to individuals across levels of experience. We will however try and discuss the various issues brought up by users across these categories.

The participants were also physically distributed across different parts of the world, with participants in France, Britain, Germany, the U.S.A and Indonesia, apart from interacting through the software, 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. It was however reported by some of the participants that the chat room did increase the sense of community allowing individuals to become more familiar with their fellow participants. The chat room also served as a source of live technical support, both for programming techniques (provided by the investigator and the other participants) and issues with the software itself (provided by the investigator).

Prior Code Sharing Experience

Participants were surveyed with regards to their code sharing experience prior to using *share*, particularly with regard to how often they borrowed code from others in creating their own work and whether they had previously made code they had written publicly available to others and the motivations and barriers in doing so.

▪All respondents previously reported borrowing code from others at some point in time [Q11], the figures below show how often they reported doing so (fig 28) and their relationship with the person creating the code (fig 29). We can see that among these participants it is fairly common for an individual to borrow code from someone that the individual does not have a personal relationship with [Q10].

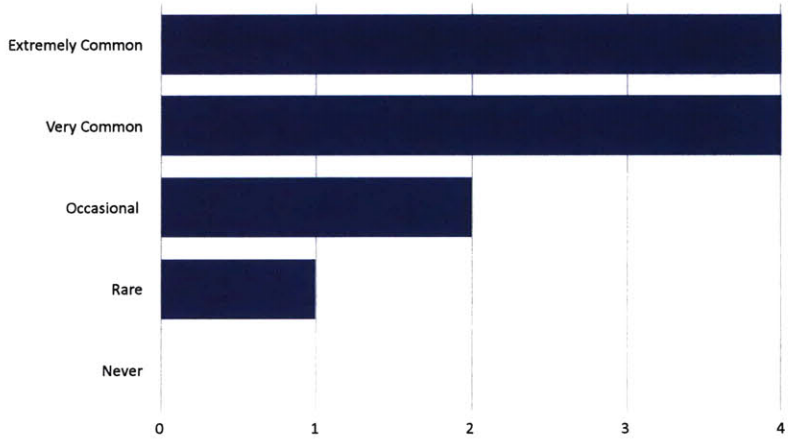

Figure 28. *Responses to question 11, "How common of an Experience is Borrowing Code".*

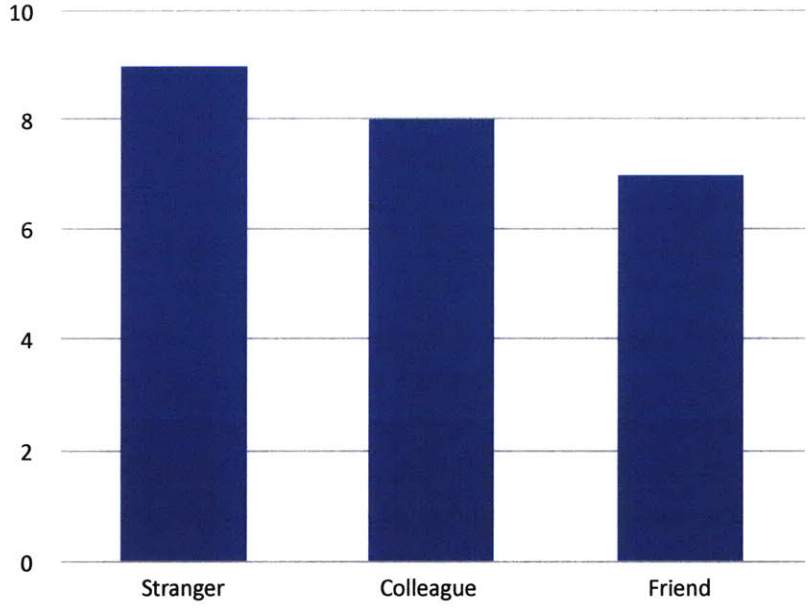

Figure 29. *Responses to question 10, "What was your relationship to the person whose code you borrowed".*

▪Seven of eleven respondents reported having made their own code publicly available (either posting it on their blog/website, releasing libraries or posting code in response to a question asked on the *processing* forums) code. Reasons participants reported for sharing their code [Q14] include:

Feeling that one ought to give back

"I have. I use a lot of open source tools, so I feel it's good karma to keep my source open as well. I have learned so much from looking at the code of others, I hope someday someone learns something from my code."

A desire to see what other people do with one's code

"Yes, I have. my motivation to do so is to let people doing experiment with my code, so that people can extend and make the sketch better. also, there might be some other publicly available people code in my sketch, I want to pass it along to other people."

Helping others solve their problems/helping the community (e.g. questions posed in the forums)

"Very rarely. if so, it was in forums where I posted parts of my own code because it was what others have been asking for."

These responses correspond with what we see in the broader open source ecosystem, with individuals sharing for 'good karma', or the pragmatic reason of getting more 'eyeballs' (and thus more improvements) onto a piece of code, or direct assistance to other member of the community.

▪When asked about barriers that prevented them from sharing their code [Q15], responses included:

Concern for 'theft'

"Afraid someone will steal my ideas. Not comfortable with someone making money off of something I give away for free."

"I don't mind sharing things. What I don't like is when a code is being used without acknowledge who created it. without saying thank you or stuff like that [...] I think issues might arise when money gets involved: for example it takes me one year to create a very good code bla bla and the next day someone makes loads of money by using it"

This worry about being 'ripped off' is a tension that exists for many when releasing software openly, whether or not one makes a living from it. We feel the issue of 'stealing' is directly related to the relationship created between the two parties when the appropriation is made, when there is attribution it is like homage, without, it is closer to plagiarism. When 'gifts' are moved from the sharing economy to a market one, it may break the norms assumed by a participant in the former and can strain the relationship between the individuals involved. This underscores the importance of creating ties (even weak ones) between members of sharing communities that provide opportunities to better negotiate the tensions that may arise with respect to these feelings.

Insecurity about code quality and publicly displaying substandard work

> *"Probably insecure about how good my code is, since I don't consider myself an expert by any means."*

> *"Bad written code! I'm self taught and had an hard time to understand [Object Oriented Programming], sharing code is like writing, and writing poorly with bad grammars and spelling is not something you want to show to everybody!"*

Interestingly, another respondent with the same concern commented that because *share* makes code sharing automatic it does not feel like one is 'releasing' software but rather it is clear from the context that you are looking at works in progress in someone's 'sketchbook'. This comment aligns with one of the design goals of *share*, to create a *good shared workspace* as opposed to an exhibition space, though it was a pleasant surprise to see that for some it actually reduced the pressure burden related to making one's work publicly visible. We hope that like the Impressionists described in chapter two, this opportunity to work 'side by side' is mutually beneficial to participants as they develop their craft.

Desire to keep certain things private

> *"I think sometimes there are ideas behind code that can be shared, but at some point, the thing that makes your project unique may not need to be shared. The key part of your code, maybe others should have to replicate it, not just copy it, and in doing so they may benefit more."*

"I don't like to share code from my artistic work."

These last two comments point to a much more complex relationship with code ownership, as stated we picked the art domain because we though it offered suitable challenges to what we were trying to do, the identity of artists that make art from code is tied intimately to that code. It is thus expected that there is an internal resistance to give it up and make one's "secret sauce" available to the world. While it is something we would be interested in investigating in future, a proper treatment of this topic would be too large an undertaking in the context of this thesis.

Other reasons included contractual agreements, for which the choice is likely out of their hands, and for others lack of a place to host their work (so while there are increasingly more options to display one's *processing* work this may suggest that it would be worth integrating some upload mechanism directly into the tools — similar to the manner in which *scratch* does)

Quantitative data

Since in order to function correctly, the software tracks the origin of all the code produced by the participants, it also provides a rich data source for quantitatively describing how code was reused over the course of the competition. Excluding sketches (projects) created by the investigator, 65 sketches were produced by the 16 participants over the two week period, 12 sketches were removed from this analysis because they either contained no code or were duplicates of another sketch created by the same participant 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, to continue working on these sketches the participants would duplicate them into new projects), thus 53 projects were used in this analysis, with each user creating an average of 3.31 projects (standard deviation=2.84, median=3), the 53 projects had a total of 154 files. The projects included 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 valiable 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; we do *not* feel that we have enough data for this to be particularly predictive over larger communities or longer time periods. It is included just to give some context in describing what happened over the two weeks.

The table below (table 2) summarizes the measures of activity encoded in the code files.

Percent content borrowed refers to the percentage of characters (not including whitespace) in projects that was borrowed from projects created by users other than the creator of that project (i.e. it excludes code that was 'borrowed' from other projects by the same author).

In degree and out degree measure the number of incoming and outgoing links to projects created by other users. An incoming link from project A to project B indicates that project B

borrowed code from project A. An outgoing link from project M to project N indicates that project M *contributed code to* project N

NUMBER OF SOURCE USERS counts the number of users from which code is borrowed in that project.

Across all Projects (53 Projects)

Percent content borrowed			13.925%	
Projects with at least 1 incoming link (i.e. borrowers)			32.075%	
Projects with at least 1 link (incoming or outgoing)			60.377%	
	Average	Std Deviation	Max	Min
In Degree	0.887	1.296	8	0
Out Degree	0.774	1.826	8	0
Total Degree	1.660	2.084	8	0
Number of source users	0.663	1.143	6	0

Table 2. *Code reuse and connection statistics across all 53 projects.*

The charts below show the distribution of some of the data above across all 16 individuals. Participants are coded P1-P16.

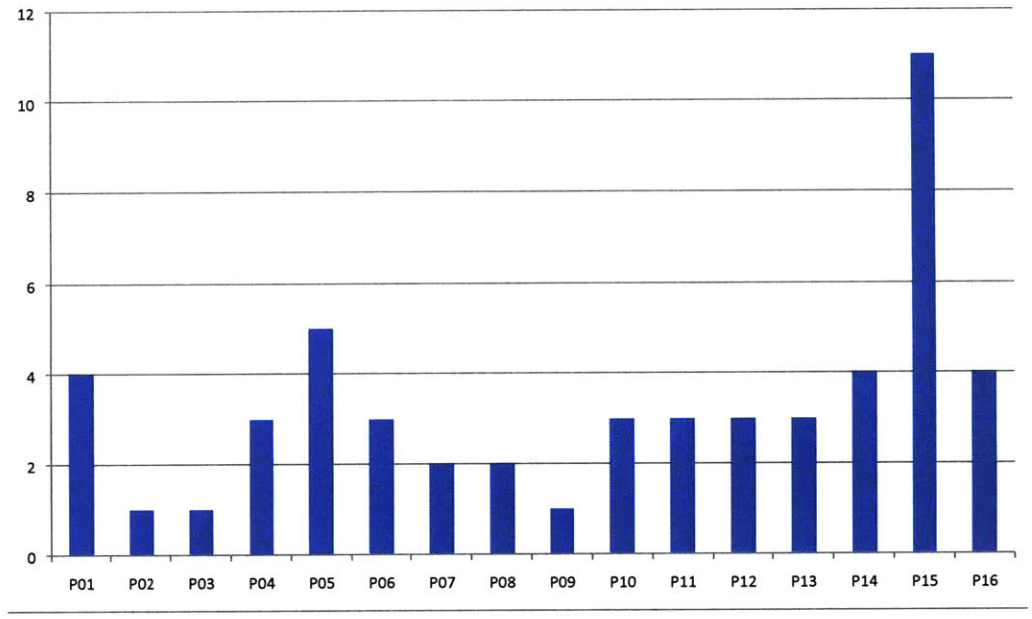

Figure 30. *Distribution of number of projects created by each user.*

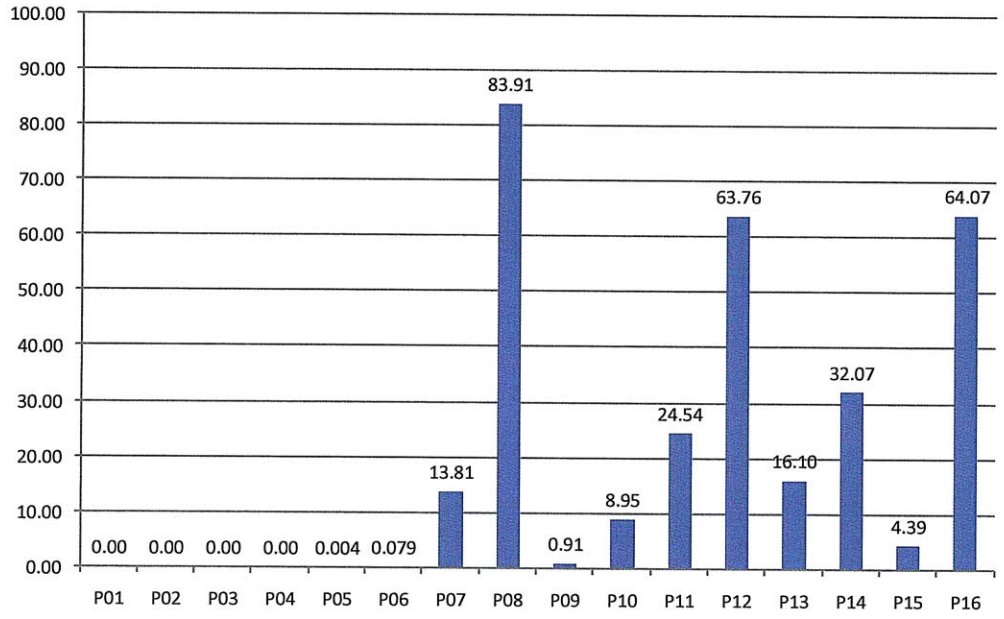

Figure 31. *Distribution of the percentage of code each user borrowed from other users.*

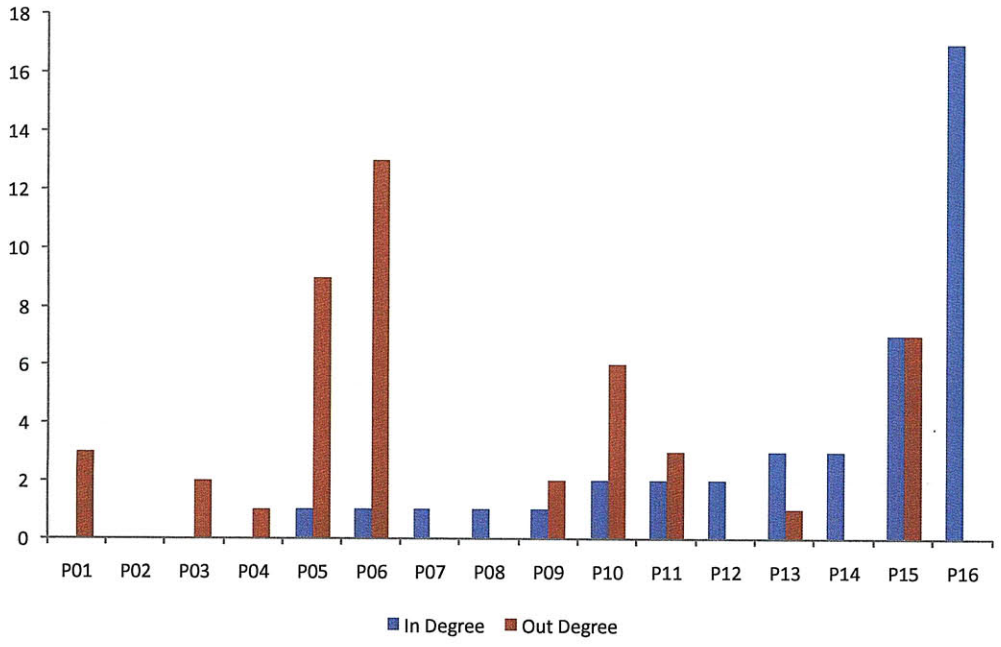

Figure 32. *Distribution of in and out degree across all projects created by a particular user.*

These graphs show that while there are quite a number of users (6) who did not borrow anyone else's code, all but one user either borrowed from or contributed to some other users' work.

We can also look at how the data on borrowed code is distributed across all 53 projects (fig 33).

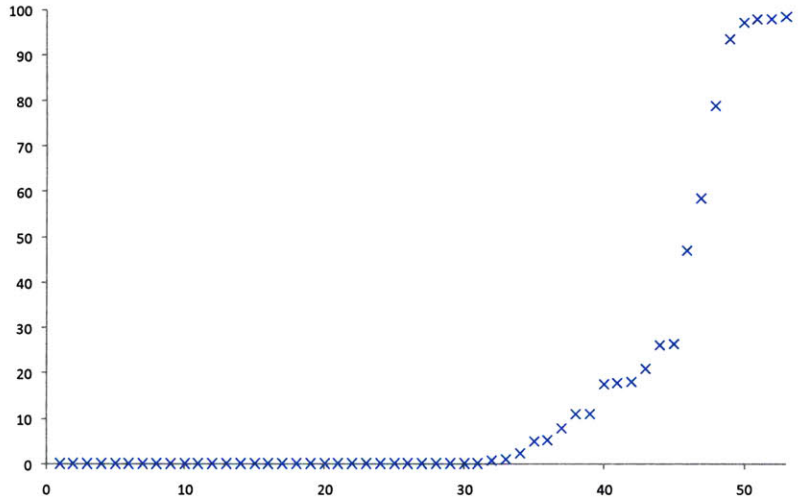

Figure 33. *Percentage of borrowed charachters across all 53 projects.*

It is also useful to look at the data among projects that have at least one incoming or outgoing link as it gives a sense of how much code is shared within the connected components of the network. While these measures overestimate the 'usefulness' of the code base (in terms of the amount code that a user was directly able to re-use) by filtering out the 'noise' associated with content that has not was not re-used by anyone, it is interesting to see the amount of activity between projects with some interdependance. 32 of the 53 projects have at least one link and form 5 connected components in the overall graph. The largest connected component contains 21 projects, with the next largest component containing 5 projects, and the remaining 3 components having 3 or fewer projects in each.

Table 3. *Code reuse and connection statistics across projects with at least one incoming or outgoing link.*

Across projects that had at least one incoming or outgoing link (32 Projects)				
Percent content borrowed			20.981%	
	Average	Std Deviation	Max	Min
In Degree	1.2813	1.464	8	0
Out Degree	1.469	2.170	6	0
Total Degree	2.75	2.048	8	1
Number of source users	1.552	1.310	6	1

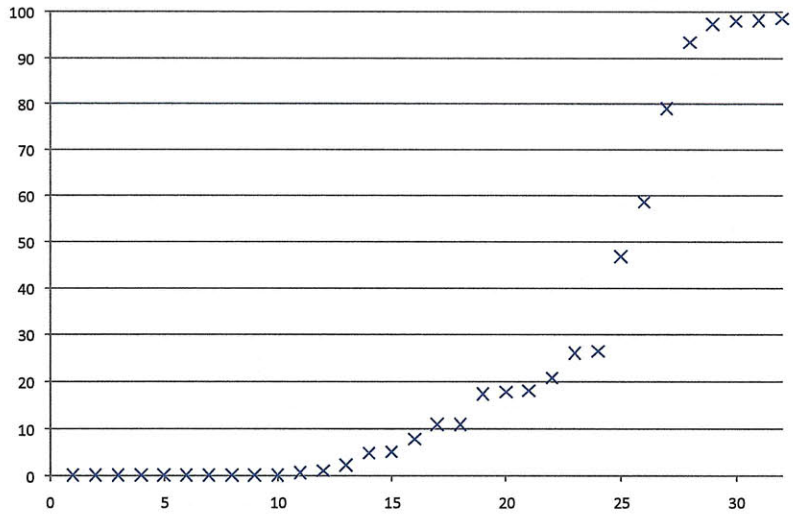

Figure 34. *Percentage of borrowed charachters across projects with at least one incoming or outgoing link.*

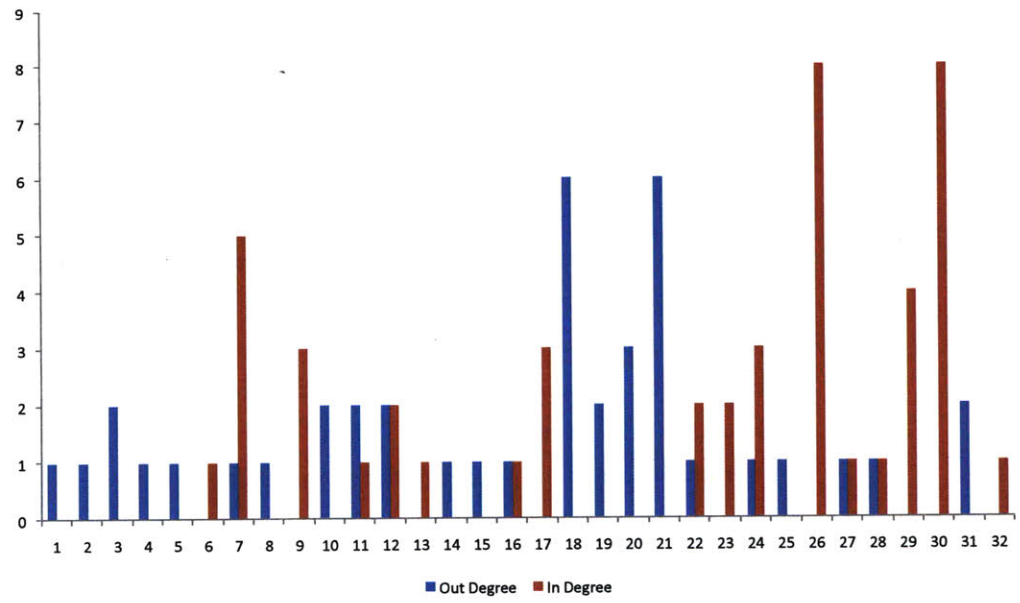

Figure 35. *Distribution of in and out degree across projects with at least one incoming or outgoing link.*

This data indicates that there was reasonable usage of the features provided by *share*. With regards to the data on amount of code borrowing it 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.

Explicit (@saw) References

The "@saw" syntax for making explicit references was not used much in the course of the competition with only nine uses across all the projects. These mainly referenced interesting ideas individuals saw in other people's code, such as using a source file to store project configuration or ideas for doing collision detection and in one case inspiration to move from a monochrome color scheme to full color.

Survey Response

Even more important than the quantitative data encoded in the software are the participants' reports of their experience using *share*. Our hypothesis was that automatic tracking and attribution of code would lower barriers to sharing code and provide encouragement to share code with others, we also were interested in what kinds of benefits individuals would receive from having others' code available to them (though those are well established in the open source literature we were curious as to whether the artistic context would add or take away from this). The data and quotes in this section come from the eleven responses to the survey that we received.

▪With regard to the value of the automatic attribution provided by *share* [Q20] 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, and also pragmatic value in seeing what had been done with it. The display of attribution also increases the sense of community (and reduces anxiety around 'stealing'). We also see that it supports something people are doing already, i.e. documenting their sources when they borrow code. *Share* makes this process seamless and also passes this information back to the original creator.

▪When asked whether the attribution features in *share* reduced barriers to sharing one's code [Q19a], responses were also quite positive.

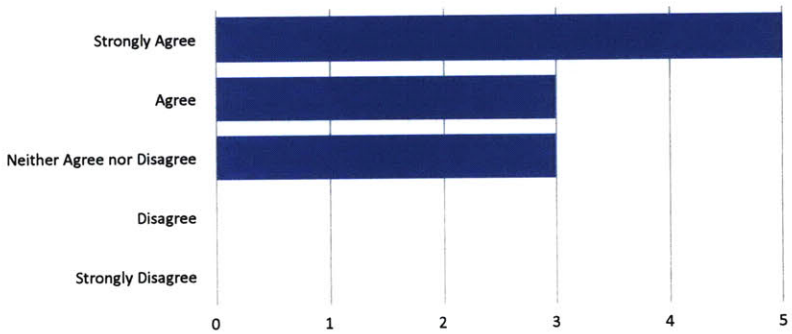

Figure 36. *Responses to question 19a, "Does share reduce your barriers to sharing code".*

▪The responses on the usefulness of being able to track your own code [Q17] largely following the answers given for [Q20] with one user saying that "It's pleasant ... it can lead to new understanding to see your code in other contexts". However not all respondents were able to comment as some had no code which was borrowed by others. The feature also became more useful as the competition progressed with a number of participants saying that it only became really useful in the "last days" of the competition.

▪With regard to using the visualization to track the movement of *other* peoples' code [Q18], 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 to [Q18] 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. We also received feedback 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. As previously mentioned in our design and implementation chapter the ability to open a project directly from the visualization had to be removed just days before the start of the competition due to conflicts between the GUI toolkit and the graphics library we used to render the visualization. However the end of the comment quoted above does point to an opportunity to encode more information about a project in the visualization itself, possibly though the use of tooltips or even by 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 because, as the number of participants was fairly low, there was very little that would remain undiscovered for a long time or that could not be discovered by browsing through the lists in the file browser. We suspect that the utility of the network visualization would increase as the size of the community using *share* grows past the number where one can reasonably keep tabs of the activities of all the members of the community in one's mind.

▪We asked participants whether they felt that the feature set in *share* made them more *productive* (able to do things more quickly) [Q19b, Q23] or more *creative* (encouraged them to do things they otherwise may not have thought of) [Q19c, Q24].

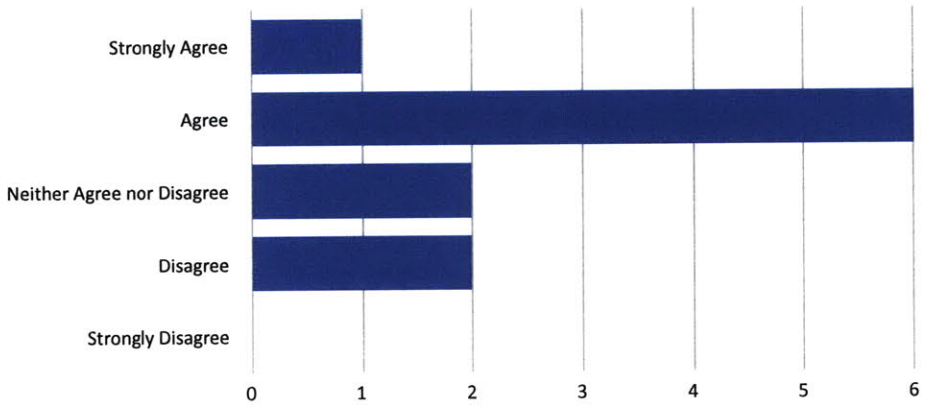

Figure 37. *Responses to question 19b, "Does share increase your ability to adress the task at hand".*

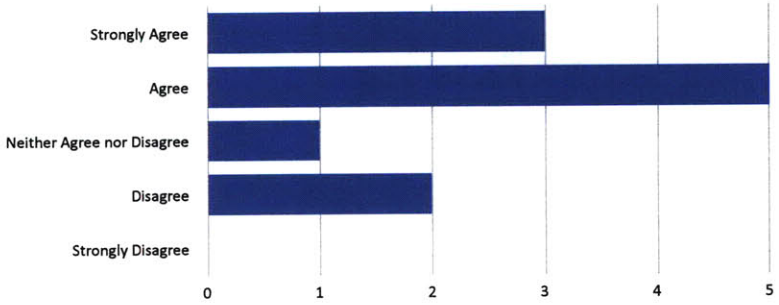

Figure 38. *Responses to question 19c, "Does share increase your creativity in addressing the task at hand".*

When asked to elaborate [Q23, Q24] 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 [similar] 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 on making it more likely that you would see something that gives you an idea; we are confident, given our observation of collaborative practice among artists as documented

in chapter two, 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."*

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 so called "user generated content" but solutions to this kind of problem (such as tagging systems) continue to be developed in numerous online spaces. We had even implemented tagging features and limited tag based browsing, but removed this feature in favor of the bookmarking feature because we knew that the size of the repository would not be so large to make this a major problem, but it is certainly something that warrants further consideration in future iterations.

▪When asked directly about disadvantages to working in *share* [Q27], all responses were related to the technical shortcomings

of the text editing component in comparison to the native *processing* editor, this was not at all surprising as our prototype is definitely in an earlier stage of development than the *processing* IDE and we also focused on the social features that we were adding as opposed to code editing features which results in a somewhat impoverished code editor. None of the users remarked on disadvantages to the concepts in *share* or even the implementation of the attribution or visualization; given the self-selection of the participants this is not completely surprising, but we are pleased that our design choices did not turn people off the idea.

▪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 [Q23] responses were as follows.

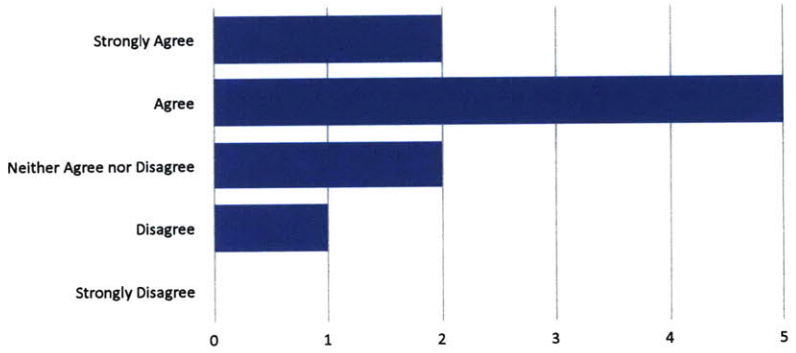

Figure 39. *Responses to question 23, "When working in share, I feel I am able to work unencumbered by the notion of working within a community".*

Users generally felt that they were able to maintain their independence.

▪When we asked participants whether they would like to continue using *share* [Q24, Q25] one respondent said that they would not use *share* in future unless it was as technically capable (and polished) as the *processing* editor, another said that they would use it as a tool for collaboration (more traditional collaboration with known collaborators). All other respondents responded that they would consider using *share* in future. ▪In response to the question on what kind of projects they would not consider working on in *share* [Q26] respondents said that they would not be able to work on commercial projects in *share* as their client relationships would likely prohibit this (or at least presumably take the decision out of their hands). But

more interestingly one user indicated that there were some projects they considered too personal or private to write in *share*, and would desire some way to mark certain sketches in *share* as private. While it is possible that this would ease the entry path to using a system like *share*, we are concerned that this would create a certain invisible asymmetry between projects and users and a decision along these lines would certainly require further consultation with the community using the tool (the current alternative suggests keeping projects you do not want public out of our software and just using the regular *processing* tool, but this does increase switching costs to our tool).

■ In summary our deployment shows that the feature set is indeed useful to users, users valued being able to see what happened to their code and valued the display of attribution that mark the use of their contribution. Users also reported that they were more productive and more creative in working on their projects due to the presence of others source code and creative output. We did not see the use of the network visualization to discover resources in the network that would have otherwise one undiscovered, however this is likely because the total number of users was small enough that one could easily keep track of what everybody else was doing. We also were not able to determine the difference in value proposition that *share* presents with respect to beginners vs. more experienced users, nor how responses to *share* differ based on whether individuals use *processing* in their professional lives.

Chapter 8. Conclusion & Future Work

This thesis 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 of programmers. Our environment facilitates the creation of a code commons available to all member of the community and allows for very fine-grained attribution for the re-appropriation of code and provides visualization, visible to all members of the community, of the patterns of sharing latent to that community. It does this by automatically sharing all the code written in it with all other users of the software, as well as tracking copy-paste operations so that it can determine the source of any text in the repository, it also provides a means of explicit attribution for types of exchange that are not captured in copy paste operations. We have highlighted the importance of social factors in motivating participation in non-market sharing economies as well as the peculiarities of motivating social exchange among very loosely associated groups of programmers that do not share a constraining project or vision to guide their

collaboration. We have also demonstrated how the design of our system uses attribution and subtle markers of reputation to address these issues of motivation while being sensitive to the individual agency desired by independent programmers.

Situated within an economy of code sharing, our original research questions were:

1. What rewards does the visualization of attribution provide to the original contributor? Do these rewards lower the barrier towards openly sharing ones code?

2. Are individuals able to track the re-appropriation of code they have contributed? If so what are the benefits to doing so?

3. Does working in such a system disrupt their regular work practice, i.e. can users program without being encumbered by the notion of participating in a community?

Our deployment with a small set of users shows great promise in the usefulness of the tool in response to these questions.

1. The automatic tracking and public display of attribution contribute to the good feelings of the participants as they feel recognized for their creative work and community contribution They also feel more at ease with reusing the work of others without feeling like they are stealing, with most of our users affirming that it reduced their barriers to sharing code.

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

3. Almost all users reported that they did not feel disrupted from their regular programming practice.

This thesis also sought to present a long-existing cultural context to support the kind of practice embodied in the design of our software, and we feel that we have provided ample evidence for the prevalence (and relevance) of re-appropriation in creative practice and the model of loosely bound cooperation both in the background work discussed and in the participants' enthusiastic response to using our software.

Future Work

We see a number of features that could be improved or added to *share* and definitely space for more research to be done. Some include technical improvements that would allow the software to scale better with increasing numbers of users. Currently *share* downloads all updated projects and their data files. While this is likely fine for the code, downloading all the attached data files could potentially consume a great deal of hard drive space. *Share* should provide a mechanism that allows for some control over the amount of disk space being used. However there are a number of issues that go beyond stabilizing the current feature set.

Design Opportunities

Scaling the visualization. With an increasing number of users and links the visualization needs to be able to scale, we think this can be done primarily by filtering the set of 'possibly interesting' nodes when we expand nodes in the building of the tree, particularly in the use case of discovery of new resources. It is not necessary for one to see everything (as end users are not interested in analyzing network structure) but rather it is better to see something that is useful (even though it may not be exhaustive)

Bootstrapping discovery. Coupled with an algorithm that could automatically suggest related resources the network visualization has the potential to be used to bootstrap discovery of useful resources for users that are currently unconnected to the rest of the network. Existing work in the analyzing code similarity and automatically retrieving related snippets such as found in the Codebroker Project (Ye, Fischer 2002) or other forms of collaborative filtering could provide mechanisms for a users code to pull related resources to the users attention.

Functional Tags. We think that a set of functional tags, that is tags whose significance the system could recognize, much like the badges given in forums like StackOverflow but applied to projects, could be used to both provide a set of useful filters but also to encourage desirable behavior. For example tags like "work-in-progress", or "open-for-feedback" could be used to filter out or provide specific views for sketches in searches or visualizations. We

arealso curious as to whether recognizing and promoting tags like "well-documented" could be leveraged to encourage behavior that is desirable to the community. We had ideas for these and a number of other functional tags and the infrastructure to implement them in *share* however we felt that the time span in the initial evaluation described in this thesis was too short to allow this behavior to emerge and be usefully documented.

Persistent Conversation Around Code. One of the challenges that we identified while developing the ideas in *share* was the separation between conversations about code (which happens on forums and irc channels) and the code itself. While it is an area we did not pursue deeply, apart from the inclusion of persistent comment threads per project, we think that it would be a rich area for further research. Either developing the IDE to better support in depth conversations around code or finding ways to integrate a system like *share* with existing online forum software, so that while reading discussions on the forums, that discussion is also available with the code.

Code Visualization. Our tool's infrastructure also provides an opportunity to look at ways to create compact visualizations of source code features. Currently projects in the visualization are represented by a color coded "document" icon, this icon shows nothing about the project itself, there may be opportunity to display, within the icon, aspects of the code itself that could be useful to those browsing, for example relative amount of documentation or relevant software metrics.

Integration with existing community tools. This is certainly something that warrants further investigation, opportunities to link to existing identity or discussion systems, like web based profile pages allows further opportunity for identity and reputation development (and display) in a manner that is portable across community sites outside of our tool.

Research Questions

Some questions also remain unanswered and new ones emerge from doing this work. We did not collect enough data in this study to report on whether there are different responses to the

features provided by *share* between those that have a commercial relationship to writing code and those who use *processing* more casually. A better understanding of how 'commercial' programmer-artists approach working in sharing vs. market economies in the context of a system like *share* may shed light on new ways that the two economies may interact in future. Our data also was not able to say very much about the different value proposition *share* provides to individuals across levels of programming experience or whether individuals in different parts of the spectrum respond to different motivational factors, larger and longer term studies would be needed to tease out these differences if they indeed exist.

Another interesting research question is whether working in a system like *share* changes how an individual writes their code, for example we imagine that it may be possible that a programmer seeing a particular piece of code they wrote becoming widely used, may be encouraged to improve the modularity of that code to make it easier for others to reuse.

Further investigation of the effect of a tool like *share* on individuals creative practice is also warranted, for example investigating the effect of our tool on the diversity of work created by a community is an interesting question.

We also see a potential for this tool in educational contexts and have received a number of inquiries from participants and visitors to our web site about the potential of using *share* in a class based settings. Examining the use of *share* in alternate settings such as classes, shared artists studios in addition to online communities is also something that we would be interested in pursuing in future.

Appendix A

This appendix contains the survey given to participants in our evaluation

Share Experiment Questionnaire

Thanks for participating in the share experiment. This questionnaire is designed to help us understand your personal experience of the project. This is part one of the questionnaire and is the core of the study. We are particularly interested in the motivations for and barriers preventing sharing ones code. We are also interested in how a tool like share may change how you work and your approach to your creative practice.

Please answer all questions to the best of your ability, every bit of information helps.

Thank you once again for your time and participation.

Biographical Information

1. Name *

First Last

2. Email *

3. Occupation *

4. Gender

Male

5. *Share* Username *

Programming Experience

Previous experience using processing or other programming languages.

6. How long have you been using processing? *

- (●) Less than 3 Months
- ( ) 3 – 6 Months
- ( ) 6 – 12 Months
- ( ) 12 – 18 Months
- ( ) More than 18 Months

7. How long have you been programming? *

- (●) Less than 3 Months
- ( ) 3 – 6 Months
- ( ) 6 – 12 Months
- ( ) 12 – 18 Months

○ More than 18 Months

7. How would you rate your experience level with processing? *

◉ Beginner ○ Intermediate ○ Advanced

8. Do you use processing in relation to your professional work? *

○ Yes ○ No

Previous Code Sharing Experience

These questions are about your experience sharing/borrowing code *before* using *share*

9. Have you ever used code from another persons work in your own projects? *

◉ Yes ○ No

10. If yes to the previous question, what was your relationship with the person(s) whose code you were using (Check all that apply)

☐ Stranger ☐ Friend ☐ Colleague

11. How common of an experience is this for you (using other peoples code)?

◉ Extremely Common

○ Very Common

○ Occasional

○ Rare

○ Never

12. Have you ever made your code publicly available to others? If so, what motivated you to do so?

13. What, if any, are the reasons that would prevent you from sharing your code publicly?

The Share Experiment

Questions about your experience using *share*

14. How long did you use *share* for (in number of days?) *

15. How useful did you find it to have other peoples' source code available to you? *

- (●) Very Useful
- ( ) Somewhat Useful
- ( ) Not Useful

16. How did you discover other peoples' code? (Check all that apply) *

- [ ] Browsing using the main window
- [ ] Searching for code
- [ ] Using the visualisation
- [ ] Did not make use of others code
- [ ] None of the above

17. Were you able to track and see what happened to code that *you* produced? Was there any benefit to being able to do this? *

18. Were you able to use the visualisation to look at the network around code *other people* produced? Was there any benefit to being able to do this? *

19. Evaluate the following statement. *

	Strongly Agree	Agree	Neither Agree nor Disagree	Disagreeq	Strongly Disagree
19a. The attribution methods provided by *share* lower the barriers I have for sharing my code	O 1	O 2	O 3	O 4	O 5
19b. The features provided by *share* increased my *ability* (made it easier) to address the task at hand.	O 1	O 2	O 3	O 4	O 5
19c. The features provided by *share* increased my *creativity* in addressing the task at hand.	O 1	O 2	O 3	O 4	O 5

20. What value to you see in the means of attribution provided by *share*? (i.e. code tracking, @saw references, network visualisation and source based highlighting)

21. How do you feel the features provided by share affected your *ability* to address the task at hand?

22. How do you feel the features provided by share affected your *creativity* in addressing the task at hand?

23. Evaluate the following statement. *

	Strongly Agree	Agree	Neither agree not disagree	Disagree	Strongly Disagree
When working in *share*, I feel I am able to work independantly and unencumbered by the notion of working within a community	O 1	O 2	O 3	O 4	O 5

Beyond the Experiment

These questions address how you would feel about using a system like *share* on a more long term basis

24. Would you consider using *share* on a more long term basis? *

● Yes O No O Other

25. If yes to previous question, what are the main reasons you would want to continue using it. If no, what are the main reasons you would not want to continue using it.

26. Are there types of projects you would not consider working in within *share*? If so what are the main reasons that would prevent you from doing so?

27. Are there any disadvantages to working in a system like share.

Bibliography

von Ahn, L., & Dabbish, L. (2004). Labeling images with a computer game. In *CHI '04: Proceedings of the 2004 conference on Human factors in computing systems* (pp. 326, 319). ACM Press. Retrieved August 8, 2009, from http://dx.doi.org/10.1145/985692.985733.

Anime Music Videos. (2006, April 25). Retrieved August 8, 2009, from http://networkedpublics.org/conference/anime_music_videos.

Apache CouchDB: The CouchDB Project. (2008). Retrieved August 8, 2009, from http://couchdb.apache.org/.

Apache Lucene - Overview. (2006). Retrieved August 8, 2009, from http://lucene.apache.org/java/docs/.

Attridge, D. (2004). *The Cambridge companion to James Joyce.* Cambridge University Press.

Bagozzi, R. P., & Dholakia, U. M. (2006). Open source software user communities: A study of participation in Linux user groups. *Management Science*, *52*(7), 1099.

Benkler, Y. (2007). *The Wealth of Networks.* Yale University Press.

Bullough, G. (1957). *Narrative and dramatic sources of Shakespeare.* New York: Columbia University Press.

Carter, S. (2001). Takeover. On *The Blueprint* [CD]. New York: Roc-A-Fella/Island Def Jam.

Carter, S. (2003). What more can I say. On *The Black Album* [CD]. New York: Roc-A-Fella/Island Def Jam.

Cox, C., & Warner, D. (2004). *Audio culture : readings in modern music.* New York: Continuum.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Sixth Symposium on Operating System Design and Implementation.* December.

Delicious. (n.d). Retrieved August 8, 2009, from http://delicious.com/.

Ding, A. (2006). *Recombinant Design: Leveraging Process Capture for Collective Creativity.* Massachusetts Institute of Technology.

Donath, J. S. (1999). Identity and deception in the virtual community. *Communities in cyberspace*, 29–59.

Fischer, G., Giaccardi, E., Eden, H., Sugimoto, M., & Ye, Y. (2005). Beyond binary choices: integrating individual and social creativity. *Int. J. Hum.-Comput. Stud.*, *63*(4-5), 482-512.

Fulford-Jones, W. (2009). Sampling. *Grove Music Online.* Retrieved August 1, 2009, from http://www.oxfordmusiconline.com/subscriber/article/grove/music/47228.

Friedman, M. (Ed.). (1982). *De Stijl, 1917-1931 : Visions of Utopia.* New York: Abbeville Press.

Ghosh, R. A. (2005). Understanding free software developers: Findings from the FLOSS study. *Perspectives on free and open source software*, 23–45.

Green, J., Thorington, H., & Riel, M. (2004, July). Networked_Performance — About. Retrieved August 8, 2009, from http://turbulence.org/blog/about/.

Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, *6*(3), 25–39.

Heer, J., Card, S. K., & Landay, J. A. (2005). Prefuse: a toolkit for

interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 421–430).

Kristeva, J. (1986). *The Kristeva reader*. New York: Columbia University Press.

von Krogh, G., & von Hippel, E. (2006). The promise of research on open source software. *Management science*, *52*(7), 975.

Lakhani, K., & Wolf, R. (n.d.). Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *Perspectives in Free and Open Source Software, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), MIT Press, Cambridge, MA.*

Lakhani, K. R., & Hippel, E. V. (2003). How open source software works: *Research Policy*, *32*(6), 923 - 943. doi: DOI: 10.1016/S0048-7333(02)00095-1.

Lave, J., & Wenger, E. (1991). *Situated Learning : Legitimate Peripheral Participation (Learning in Doing: Social, Cognitive & Computational Perspectives)*. Cambridge University Press. Retrieved August 8, 2009, from http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0521423740.

Lenz v. Universal | Electronic Frontier Foundation. (n.d.). . Retrieved August 9, 2009, from http://www.eff.org/cases/lenz-v-universal.

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. *Journal of Law, Economics, and Organization*, *21*(1), 20–56.

Lessig, L. (2008). *Remix : making art and commerce thrive in the hybrid economy*. New York: Penguin Press.

Lethem, J. (2007). The ecstasy of influence: A plagiarism, (Harper's Magazine). Retrieved July 3, 2009, from http://www.harpers.org/archive/2007/02/0081387.

Licenses - Creative Commons. (2009). Retrieved August 8, 2009, from http://creativecommons.org/about/licenses/.

Maar, M. (2005). *The Two Lolitas*. Verso Books.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: a sneak preview. In *Creating, Connecting and Collaborating through Computing, 2004. Proceedings. Second International Conference on* (pp. 104-109). Presented at the Creating, Connecting and Collaborating through Computing,

2004. Proceedings. Second International Conference on. doi: 10.1109/C5.2004.1314376.

Manovich, L. (2005). *Remix and remixability.* Retrieved July 27, 2009, from http://rhizome.org/discuss/view/19303#36752.

Manovich, L. (2001). *The language of new media.* The MIT Press.

Manovich, L. (2007). *What Comes After Remix?* Retrieved July 27, 2009, from http://manovich.net/DOCS/remix_2007_2.doc.

Metz, G. (1989). *Sources of four plays ascribed to Shakespeare.* Columbia: University of Missouri Press.

Monroy-Hernández, A. (n.d.). *ScratchR : a platform for sharing user-generated programmable media.* Thesis, . Retrieved August 8, 2009, from http://dspace.mit.edu/handle/1721.1/42977

Muir, K. (1978). *The sources of Shakespeare's plays.* New Haven: Yale University Press.

OPENSTUDIO. (n.d). Retrieved August 8, 2009, from http://openstudio.media.mit.edu/.

Oswald, J. (1990, October 2). plunderphonics press release. Retrieved July 9, 2009, from http://www.plunderphonics.com/xhtml/xnegation.html.

Oswald, J. (1985). Plunderphonics, or Audio Piracy as a Compositional Prerogative. Retrieved July 9, 2009, from http://www.plunderphonics.com/xhtml/xplunder.html.

Overy, P. (1969). *De Stijl.* London: Studio Vista.

Raymond, E. S. (n.d.). *The cathedral and the bazaar, 1997.* Retrieved from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/.

Reas, C., & Fry, B. (2007). *Processing: A Programming Handbook for Visual Designers and Artists.* The MIT Press.

Reese, B. (2005). *I'm a hustla.* On *I'm a hustla* [CD]. Full Surface/Ruff Ryders/J Records.

Stack Overflow. (2009). Retrieved August 8, 2009, from http://stackoverflow.com/.

Stallman, R. (1998). The GNU Project. Retrieved October 27, 2008, from http://www.gnu.org/gnu/thegnuproject.html.

Stanford Copyright & Fair Use - Summaries of Fair Use Cases. (n.d.). . Retrieved August 12, 2009, from http://fairuse.stanford.edu/Copyright_and_Fair_Use_Overview/chapter9/9-c.html.

Stanford Copyright & Fair Use Center. (2005). . Retrieved August 9, 2009, from http://fairuse.stanford.edu/index.html.

The Competitive Spectrum - Yahoo! Design Pattern Library. (n.d.). . Retrieved July 28, 2009, from http://developer.yahoo.com/ypatterns/pattern.php?pattern=competitive.

The Competitive Spectrum Pattern - Yahoo! Design Pattern Library. (n.d.). . Retrieved August 12, 2009, from http://developer.yahoo.com/ypatterns/pattern.php?pattern=competitive.

The Free Software Definition. (n.d.). Retrieved July 11, 2009, from http://www.gnu.org/philosophy/free-sw.html#exportcontrol.

The GNU General Public License. (2007, June 29). . Retrieved August 8, 2009, from http://www.gnu.org/copyleft/gpl.html.

Torvalds, L., & Diamond, D. (2002). *Just for Fun*. Harper Collins.

Torvalds, L. (1991). What would you like to see most in minix? - comp.os.minix | Google Groups. Retrieved July 21, 2009, from http://groups.google.com/group/comp.os.minix/msg/b5fb8c0380d0edae.

U.S. Copyright Office - Copyright Law. (n.d.). Retrieved July 9, 2009, from http://www.copyright.gov/title17/92chap1.html#106.

U.S. Copyright Office - Fair Use. (2009). Retrieved August 9, 2009, from http://www.copyright.gov/fls/fl102.html.

United States Copyright Office A Brief Introduction and History. (2009). Retrieved July 8, 2009, from http://www.copyright.gov/circs/circ1a.html.

Wenger, E. (, n.d). Communities of practice. Retrieved October 25, 2008, from http://www.ewenger.com/theory/index.htm.

White, B. (1996). *Impressionists side by side : their friendships, rivalries, and artistic exchanges* (1st ed.). New York: Random House Inc.

Yahoo! Design Pattern Library - Reputation. (n.d.). Retrieved July 28, 2009, from http://developer.yahoo.com/ypatterns/parent.php?pattern=reputation.

Ye, Y., & Fischer, G. (2002). Information delivery in support of

learning reusable software components on demand. In *Proceedings of the 7th international conference on Intelligent user interfaces* (pp. 159-166). ACM Press New York, NY, USA.

YouTube - Broadcast Yourself. (2005, February 15). Retrieved August 8, 2009, from http://www.youtube.com/.