**Dissertation Proposal:**

# Ideology and Cultural Values in the NeXT/Apple "Cocoa" Software Developer Community: A History and Ethnography

Hansen Hsu
Cornell University, Science & Technology Studies

Dissertations in Progress Session, SIGCIS Workshop, SHOT 2010 Meeting

## Background and Research Questions

This study proposes to be a social history and ethnographic study of the "Cocoa" software developer community. "Cocoa" is the name of Apple's object-oriented software platform at the heart of the Mac OS X desktop computing operating system, as well as iOS, the operating system powering the iPhone, iPad, and iPod Touch mobile devices. The ability for third parties to develop applications, or "Apps" for iOS devices using the Cocoa technologies has helped to drive the popularity of these devices. However, the community of third party developers with expertise in Cocoa predates the iPhone, and traces its history back to the 1990s. The Cocoa community is driven by cultural and technical values that shape the design of software Apps. As Apps are the primary interface between users and Apple's devices, the values embedded in them are consequential for millions of users.

Since its release in 2007, Apple's iPhone device has been extremely successful. Culturally, the iPhone has become a status symbol and a cult brand, garnering such appellations as "the Jesus phone." (Belk & Tumbat, 2005; Campbell & La Pastina, 2010) Recent statistics show the iPhone garnering 28% of smartphone marketshare.[1] Since its debut the iPhone has helped transform the mobile phone market as competitors have emulated Apple in releasing smartphones with similar capabilities, form factors, and user interfaces. The iPhone's most significant feature since 2008 has been the creation of a vibrant marketplace for third party applications. With over 250,000 "Apps" in Apple's App Store compared to less than 100,000 written for Google's competing Android smartphone operating system (Kane & Catan, 2010), despite the eclipse of Apple's overall marketshare by Android-powered devices, the availability of Apps has become one of the major advantages of Apple's iOS platform, which Google, Research In Motion, and other competitors have struggled to match. The explosion of Apps for the iPhone since 2008 was enabled in part by Apple's "Cocoa" software development technology and by initial low barriers to entry and Apple policies that evened the playing field, allowing individual developers and even amateurs to initially compete with large corporations. Investors, entrepreneurs, and programmers seeking independence flocked to create or invest in iPhone software, resulting in a "gold rush" likened to the 2000 tech bubble. If the third party applications market is major source of the value of the iPhone versus its competitors, then the developers of those applications exert significant influence over the experiences of

[1] Smartphones are mobile phones designed to have more sophisticated computing capabilities than standard mobile phones, including e-mail and web access. The marketshare leader currently remains Research In Motion's Blackberry, with 35%. (Kellog, 2010)

iPhone users. Although the "gold rush" has seen a rapid influx of new programmers to the platform, the core set of the community, which, along with Apple, upholds the cultural and technical values associated with the "Cocoa" technology, consists of developers who have cultivated expertise in Cocoa since its early days as "NextStep."

Cocoa and Mac OS X both trace their history to NeXT, the startup that Steven P. Jobs, co-founder of Apple, formed in 1985 after he was ousted from Apple's board. In 1997, Jobs returned to the company when it acquired NeXT in order to replace Apple's original, aging Macintosh operating system. NeXT had developed an innovative operating system called "NextStep," whose most valuable feature was its object-oriented software development environment. In the 1980s, object-oriented programming (OOP) was an emerging software engineering paradigm within the software industry, having previously been promoted within academic computer science. In this new paradigm, data and the actions that operate on them (called methods, analogous to a "subroutine" in traditional procedural languages) are black-boxed in "objects", in order to protect the data from accidental or malicious alteration. These objects performed actions by sending messages to each other, and a system called "dynamic binding" allowed the same message, sent to different objects, to invoke different behaviors. This allowed a model of software building that was less like "writing code" in the traditional sense, and more like building something out of a kit of Lego bricks. Many of the ideas in the dynamic OOP model had originated in Alan Kay's Smalltalk language at Xerox PARC in the 1970s, where famously Steve Jobs acquired the idea of the graphical user interface for the Macintosh. In the 1980s and '90s, OO advocate Brad Cox promoted the notion that OOP would usher in a "software industrial revolution," solving the "software crisis" that had plagued the industry since the 1960s by creating a market for interchangeable objects that could be purchased off-the-shelf and easily plugged into one's own software.(Cox, 1990a, 1990b; Ensmenger & Aspray, 2002) Though popular with academics and finding a small niche in banking and custom software development, Smalltalk's adoption in industry had been hampered by a lack of compatibility with most existing systems, so Cox created a new, hybrid language called "Objective-C," marrying Smalltalk's OO features with procedural C, one of the most popular languages in the industry. NeXT adopted Objective-C as the basis for its object-oriented "framework", or software development platform. After Apple acquired NeXT and made NextStep the basis for Mac OS X, it renamed this framework "Cocoa," to differentiate it from the development environment that maintained compatibility with older Macintosh software, the procedural and C-based "Carbon" framework. A decade later, Cocoa became the basis for the development environment of both the iPhone and the iPad.

NextStep, like Smalltalk before it, only managed to find a niche market in custom rapid-application development in Fortune 500 corporations and universities. The core of the current iPhone and Mac OS X software development community can be traced to the small community of NeXT developers in the 1990s. Most of these developers worked as freelance contractors for the large institutions that hired them; some worked in banking and had previously been Smalltalk programmers. What drew these developers to what was then, by most standards, an unsuccessful platform, was a belief in the technological promise of the OO paradigm, and a belief that OO served to make the developer's experience more productive and enjoyable by eliminating repetitive tasks, which both wasted their time and were sources of bugs. OO also allowed for software, particularly applications with graphical user interfaces, to be more flexible and thus better respond to the needs of end users. NeXT

developers considered NextStep to be technologically superior to the most popular platforms in the industry, and represented the direction the industry ought to take, but doomed to languish in a marginal niche. Thus, in the early 1990s, NeXTStep had achieved its own cult status among a tiny cadre of software developers, with similar minority status as Apple's much larger Macintosh platform (for more on the "Cult of Mac," see (Belk & Tumbat, 2005; Kahney, 2004)). Upon its acquisition by Apple and the return of NeXT CEO Jobs to Apple's leadership, the cult status of NeXT and of the Macintosh began to merge. The NeXT community welcomed the change as the Macintosh promised a large new market for them in consumer applications. In the 2000s this community achieved modest success as independent ("indie") software developers, creating and marketing their own small-scale, original applications through the Internet. (van Meeteren, 2008) The advent of the iPhone App Store has created huge opportunities as well as some anxieties within the community that the large influx of developers from other platforms may dilute the values and cohesion of the community.

My historical interest involves how the NeXT developer community became the Mac "Cocoa" developer community after the Apple acquisition and the release of Mac OS X. My argument is that the norms and values driving the Cocoa community are a result of the convergence of NeXT developer values and Macintosh user values. NeXT values are associated with the asserted technological superiority of object-oriented programming using dynamic binding, the specific implementation of these ideas in Cocoa using the hybrid language Objective-C, and the tight integration of Cocoa with the operating system, developer tools, and graphical user interface, as manifested in the graphical programming tool, Interface Builder. Mac values assert ease of use and aesthetically pleasing while functional design of both hardware and software as goals for engineers and software developers. While to some extent there was some degree of overlap in these values, particularly because Steve Jobs founded both companies and made graphical user interfaces features of both platforms, the NeXT community was focused more on ease of use and programming for developers, while the classic Mac community put up with difficult tools and programming systems as long as the end goal better served the user. The tight-coupling of graphical user interface and object-oriented programming on NextStep, and on Smalltalk upon which it was based, meant that even prior to the acquisition, arguments were made that dynamic object-oriented systems better served the user by allowing developers to more easily produce graphical user interface applications. However NeXT's market for its software was not consumer end-users, like Apple's Macintosh, but large corporate and academic institutions with low-volume needs. Thus, the norm that software should be easy to use for ordinary people was less of a driving force for NeXT developers than Mac developers. In addition, the original Macintosh programming environment was procedural (being based originally on Pascal), thus divorcing the tight coupling of the tools and the interface that had obtained in Smalltalk. After Apple's acquisition of NeXT and the replacement of the original Mac OS with the NextStep-derived Mac OS X, and the transformation of NeXT software contractors into Mac consumer software developers, these two sets of norms began to merge. This project hopes to explicate and follow this transformation within the community.

In the present, the "Cocoa" developer community might in some respects be viewed as the cultural and technological vanguard of the Apple fan and user community at large. Much of the ideology and cultural values associated with Apple products originate with Apple itself, and especially with Steve Jobs. However, Cocoa developers see themselves as

partners with Apple in promoting Apple's software platforms and their associated values. Moreover, there is a revolving door between the third party Cocoa community and Apple itself, with Apple frequently hiring well-known developers and many former Apple employees striking out for themselves. Thus to some extent a study of the Cocoa community can be seen as proxy for a study of the cultural values operating among Apple software engineers, who are difficult to get access to. The Cocoa community is widely dispersed geographically and maintains cohesion through an online presence in blogs, mailing lists, and Twitter correspondence. Cocoa developers support Apple's goals in general, but will take it upon themselves to criticize Apple when the company acts against their interests or the interests of users, or in a way that they believe is contrary to Apple's professed values and the greater good of Apple's platforms.(van Meeteren, 2008)

While Michiel van Meeteren's work has accurately described the norms of collegiality and aesthetic craftsmanship at work within the Cocoa community, I aim to extend his work both by conducting a more extensive qualitative case study of a primary site within the community and by tracing its history. My research aims to combine a social history of the NeXT/Cocoa software developer community from the late 1980s through the present, and an ethnographic study of a particular node in this community centered in Seattle, Washington. It is also an enquiry on the role that ideology and cultural values play within a community of software developers. Who were the first NeXT developers, and how did the community adjust and adapt to the changes in the market and their primary patron over the last two decades? How were the ideologies of NeXT and Apple integrated after the acquisition, and how did community members navigate tensions between the two?

The purpose of the ethnographic component of the project is to explore the culture, values, and ideological commitments of Cocoa developers, to explain their drives and motivations in supporting and programming for Apple's technology, and to understand what being a Cocoa programmer means to their lives. What motivates their passionate commitment to what they consider to be a superior technology? What draws them to the platform, and how are they enrolled as members into the community and its values? What role does the actors' notions of "technological superiority" and "quality" play in their view of their work, identity, and their efforts to "evangelize" the technology? How do they navigate a sometimes positive, sometimes tendentious relationship with Apple, to whom their livelihood is tied? How do their commitments and values affect design decisions that go into the construction of their products? How have they policed the boundaries of the Cocoa community, and how do they attempt to maintain and spread their core values in the midst of a rapid expansion of developers on the platform?

**Theoretical and Analytical Categories**

There may be some question as to why I believe the term "ideology" properly applies to the system of values that motivate Cocoa developers. One problem is that there are many different definitions of the term, and it is difficult to be precise about what one means by ideology. Raymond Williams lays out three main definitions: 1) a system of beliefs characteristic of a social group or class; 2) a system of false or illusory beliefs; 3) the (cultural or symbolic) process of production of meaning and values. Out of these three definitions, mine is closest to the third, in line with Clifford Geertz's view of ideology as a cultural system. (Geertz, 1973a) It might also be accurate to say that the first definition also applies, although only in the sense of a particular social group (the Cocoa community), rather than of

a class; although Cocoa developers tend to be of middle or upper middle class origin, in order to take their values seriously, I am not interested in reducing them to expressions of bourgeois interests. For the same reasons I am not interested in a pejorative definition of ideology as illusion or false consciousness from an analytical perspective, although such a view from an actor's perspective might be revealing. Of the sixteen different definitions listed by Terry Eagleton, the applicable ones to this study would also include, in addition to the two already listed by Raymond, "that which offers a position for a subject", "identity thinking", "the medium in which conscious social actors make sense of their world", "action-oriented sets of beliefs" and "the process whereby social life is converted to a natural reality."(Eagleton, 1991, pp. 1-2) I am thus interested in a definition of ideology that encompasses the cultural system of values that motivate and drive action for Cocoa developers, govern their technical practice, and serve to mark and produce their identities as Cocoa developers. In this I follow Althusser in viewing ideology as a system of legitimation that helps interpolate an individual's subject position, although I reject his view that such systems, and the apparatuses which transmit and reproduce them, ultimately reduce to political and economic domination. Rather I am interested in how such moral values are translated into technical ones.

Campbell and La Pastina, Belk and Tumbat, and Kahney focus on the popular perception of Macintosh and Apple fandom as a religion. While viewing Apple fandom as a religion metaphorically is useful for these authors, few would argue that it is actually a literal religion. But Geertz and others have noted the similarities between the concepts of ideology and religion, (Geertz, 1973b, pp. 199-200); both may exhibit militancy, fanaticism, and dogma. As a secular set of beliefs, "ideology" is more appropriate a category than "religion" with which to study Cocoa community values.

Hugh Gusterson has written about how nuclear weapons scientists of various political alignments become enrolled in a pro-weapons moral ideology through rituals of security background checking and weapons testing, and how their identities are reinforced and enacted through practices of secrecy that make them feel as if part of a select elite. (Gusterson, 1996) An analogous culture of secrecy exists within Apple, although within the third party Cocoa community the value of information sharing is more prevalent. However, the community's long years as a small niche, and its belief in the superiority of its technical tools and practices, do elicit within Cocoa developers a sense of being a better-informed elite among programmers; as in Gusterson's account, this project will investigate how ideology is manifested in the everyday technical practice of these developers, and how they educate newcomers into their community.

A number of other conceptual categories might also have been used in place of "ideology." Paul Edwards has used the Foucauldian concept of "discourse" in his study of Cold War "closed world" and "cyborg" political and metaphorical frames that motivated computer research in the twentieth century (Edwards, 1996). "Discourse" is extremely useful for analyzing tropes, metaphors, and figurative or symbolic representations of the Steve Jobs, the Macintosh, its loyal developers and its antagonists (Microsoft) within a mythic hero narrative.(Belk & Tumbat, 2005) This is particularly useful for examining the cultural mechanisms at work within the Cocoa community. However, the term "discourse" is less useful for describing a system of beliefs and values with moral or political import for practitioners. While the articulation of these cultural values in technical terms could be

described as "discursive," the notion has less application than "ideology" to how individuals are enrolled in a normative project upon entering a community, and how their identities are co-constructed in this process.

Other relevant concepts to this project are "paradigm" and "technological frame." Kuhn's notion of paradigm has the useful component of a technical worldview that is conceptually incommensurate with others, simultaneously theoretical and social, which takes over in revolutionary transformations of a field. However Kuhn's concept as originally articulated is tied too closely to scientific research rather than engineering practice, with its components of exemplars and anomalies. Moreover, as Paul Edwards has mentioned, the term has been "popularized to the point of vulgarity."(Edwards, 1996, p. 32) This popularization has led to the term becoming rather an actor's category; object-oriented programming is commonly referred to as being a different paradigm than its procedural predecessor, and thus when I use the term, I will use it in this actor's sense rather than in Kuhn's more specific sense. Related to the notion of paradigm is Wiebe Bijker's "technological frame." Both because this term more specifically refers to technology and because it is known only within the STS literature, it is more useful as an analytic concept. Like a paradigm, a technological frame is organized around solving particular problems that have been articulated as being centrally important by the community associated with it, and these priorities are articulated in culturally specific ways. Engineers may have high or low "inclusion" within a particular frame, and different levels of "inclusion" within multiple frame, which, like paradigms, may struggle against each other for dominance. These features of the concept are extremely useful and thus I may refer to the concept occasionally; however in general "ideology" is more pertinent to this project as the notion of "technological frame" is primarily about the technical priorities within a given problem domain, and not about the moral cultural values associated with a technical community.

A number of recent historical and/or ethnographic studies of computing have looked into the cultural values and ideology of engineers or software developers (Turner, 2006; Coleman, 2004, 2009; Malaby, 2009) Turner explores the transformation of cybernetic countercultural values into computer libertarian values from the 1960s to the 1990s. Steve Jobs and Apple have been at the center of the countercultural computer narrative, and it is appropriate to explore the ideology of Apple's developers in connection with these earlier projects. Turner noted that one aspect of the commune movement was its contradictory political stance against agonistic politics; Malaby and Coleman have shown similar (a)political attitudes within the developers of Second Life and in the free and open source developer community, respectively. Coleman's study of the free software community reveals that, like the Cocoa developer community, free software constitutes an ideology advocating specific moral values through technical practice. My project will contribute to this growing literature on computing and ideology.

One additional note about my use of the term "platform." In the computer industry, "platform" is an actor's category. Tarleton Gillespie has identified in the Oxford English Dictionary four broad categories of usage of the word "platform": architectural, figurative, political, and computational. He describes the computational meaning as "an infrastructure that supports the design and use of particular applications, be it computer hardware, operating systems, gaming devices, mobile devices, digital disc formats… The term has also been used to describe the online environments that allow users to design and deploy

applications they design or that are offered by third parties… [often by making public APIs (application programming interfaces)]" (Gillespie, 2010, p. 3) The Macintosh versus the Windows PC is the quintessential platform war, but others include Facebook versus MySpace, and the iPhone/iOS versus Android. "Platform" is thus a conveniently general term that can encompass hardware or software, as long as there is the sense that third parties can build upon it. Platforms matter because of network effects (technological momentum) (Hughes, 1987), which give their controllers market power. Part of this inertia is because technical platforms, whether hardware or software, have obduracy–it can be a significant undertaking to port a piece of software from one platform to another, often requiring rewriting. Gillespie also notes that the word "platform" suggests "neutrality towards use– …flat, featureless, and open to all… A computing platform can be agnostic about what you might want to do with it, but either neutral ('cross-platform') or very much not neutral ('platform-dependent') to which provider's application you'd like to use." (Gillespie, 2010, p. 4)

An alternate definition of "platform" within STS can be found in (Keating & Cambrosio, 2000, 2003) Keating and Cambrosio's definition of "biomedical platform" is an analytical category derived from actors' "native" use of the term in biomedicine, and seems to refer to a heterogeneous configuration of tools/assays, practices, definitions, standards and conventions, which constitute an infrastructure upon which multiple biomedical fields may rest. Keating and Cambrosio note that their notion of "platform" does not exactly fit the definition of infrastructure given by (Bowker & Star, 1999) because platforms are visible while infrastructure is typically invisible until breakdown. They also note that "platforms are active, generative… As opposed to infrastructures that show or are supposed to show some sort of historical continuity, platforms are made for contingencies: they are only for the time being." (Keating & Cambrosio, 2000, p. 359) Computer software platforms are like and unlike both infrastructures and biomedical platforms–they may be visible or invisible, transparent or opaque. However a computer platform is much more like infrastructure than a biomedical platform due to its foundational role for the software that runs on top of it. More than just conventions and standards, there is a level of "materiality" or at least obduracy in software code (boyd, 2006; Lessig, 2006) that make certain things possible or easy and others difficult or impossible if the platform producers (Apple in this case) have not provided API (application programmer interface) calls or hooks to allow developers to do certain things. Although in principle software is infinitely malleable, in practice, required continuity with installed user bases mean that software platforms are much more beholden to history than Keating and Cambrosio's definition would have it. A computer platform, as used by "natives" in computing as opposed to biomedicine, is also not a hetereogenous configuration of tools, practices, and conventions, but rather a stable and interdependent stack of hardware artifacts and software code layered upon each other. Keating and Cambriosio's definition of platform is a bit too specific to biomedicine to be analytically applicable here.

**Intellectual Merit**

This dissertation will contribute to the growing literature in history of computing devoted to software. The late Michael Mahoney noted that the history of computing has focused primarily on hardware artifacts (Mahoney, 2004, 2005, 2008). His call for historical studies of software has led to a growing body of work in this area (Agalianos, Whitty, &

Noss, 2006; Campbell-Kelly, 2003; Ensmenger, 2009, 2010; Ensmenger & Aspray, 2002; Haigh, 2002, 2006, 2009; Mahoney, 2002a, 2002b). Prior to this recent scholarship, the literature on software history had been heavily dominated by participant accounts and their preoccupations, with the result that the literature had bifurcated into internalist histories of programming languages and systems software, and business histories of software companies, with an undue focus on the PC software industry, which was most visible in the media. The recent literature has made some progress towards bringing the history of software more in line with the history of technology. These works are varied: Mahoney tries to relate the history of software as a whole to the history of automation; Ensmenger looks at it from the perspective of labor and professionalization; Haigh focuses on specific software industries such as databases and word processing; Agalianos et. al. look at the social history of Logo, as both a programming language and a radical educational philosophy; Campbell-Kelly looks at the history of the industry as a whole, but reorients its study towards the contracting and corporate software sectors, which have been largely invisible in the literature but made up the majority of the industry from the 1950s through the mid-1970s, and still account for a vast share of the industry. Tom Haigh's work and others have taken up Mahoney's call towards the study of software applications. Mahoney said that "what makes the history of software hard" is that it is really the history of the communities of practice that model their world into symbolic code.(Mahoney, 2008) The software as archive or as artifact are both extremely difficult texts to read because only through dynamic interaction with the running program, rather than reading the static program text, can one get an intimate sense of it, yet the problems of obsolescence of hardware, and inadequacy of documentation make historians' access to legacy software problematic. Yet fortunately, although software itself may be difficult to get at, the communities that produced them, first and foremost the programmers, are mostly still alive and available to us. For this reason, a turn towards social history of software developer communities is in order. Although though much has been written about the early PC hobbyist community (Freiberger & Swaine, 2000; Ceruzzi, 1996, 1998; Campbell-Kelly & Aspray, 1996; Turner, 2006; Markoff, 2005; Levy, 2001), only Akera's work on the IBM SHARE user group (Akera, 2001, 2007) specifically explores the history of a software developer community, especially one devoted to a particular platform tied to a specific manufacturer. Mahoney also understood that new programming language paradigms such as object oriented programming were simultaneously technical and managerial, and were designed to foster good practices, or in other words, "proper ways to think about programming"(Mahoney, 2008, p. 13) among practitioners. My dissertation is interested in interrogating how these engineering and managerial concerns dovetail with the influence of more macrosocial countercultural and cybernetic values that might have played a part in the development of PCs and graphical user interfaces (Bardini, 2000; Bardini & Horvath, 1995; Kay, 1993; Turner, 2006). In doing this, the older history of programming languages and systems software is revised by examining them in terms of community practices and norms that embed and reproduce social relations. By explaining how a particular community of software developers translates its members' work practice, concerns, and values into computational microworlds, it is my hope that this will be a contribution to Mahoney's call for works viewing "software as model, software as experience, software as medium of thought and action, software as environment within which people work and live." (Mahoney, 2008, p. 12)

Whereas the history of software is only beginning to look at communities of programmers, ethnographies of software, unsurprisingly, have made them their primary

focus. A majority of these have focused on the free and open source software (FOSS) communities. (Coleman, 2004, 2009; Kelty, 2008; Malaby, 2009). Many of these are concerned with the political resistance against intellectual property. Malaby's work looks at a specific software company, the developers of the online world, Second Life. These works have highlighted the role of values and ideologies, specifically the so-called "hacker ethic,"(Levy, 2001) governing these subcultures within software development. This study will provide a counterpoint to these studies by exploring the role of cultural values within a software community tied to a monopolistic and proprietary manufacturer, which values information sharing while reconciling it with entrepreneurial profit-seeking.

There is an extensive popular literature on the history of the first PCs, Xerox PARC, Apple, the Macintosh, and the life of Steve Jobs (references). Critical scholarship is needed to correct some of the myth-making aspects of these works. Even Sherry Turkle, in *Life on the Screen* (Turkle 1995), wrote that graphical interface Macintosh versus command-line DOS PC users worked in psychologically different modes, the former embracing a "culture of simulation" where transparency meant the user knowing how to get the computer to do something for her, and the latter celebrating a "culture of simulation," where transparency meant access to technical underpinnings. This essentialism only reifies the popular narratives about the Macintosh. Some recent work (Belk & Tumbat, 2005; Campbell & La Pastina, 2010; van Meeteren, 2008) has begun to address some of these problems. None, however, do so from an STS perspective.

The canon of constructivist technology studies has long valued the role of users (Bijker, 1995; Kline & Pinch, 1996; Pinch & Bijker, 1984), but recent works have focused more explicitly on "the user" as an analytical category (Akrich, 1992; Cowan, 1987; Greenberg, 2008; Laegran, 2003; Lindsay, 2003; Oudshoorn & Pinch, 2003; Woolgar, 1991; Wyatt, 2003). Many of these works (Woolgar 1991, Lindsay 2003, and Laegran 2003 in particular) are case studies of computing, as the category of "the user" has been important for user interface designers within the field of human-computer interaction (HCI) (Cooper & Bowers, 1995). Although in many respects, software developers are producers and "manufacturers" of technology rather than users, in the case of many small-scale independent iPhone and Cocoa developers, they have more in common with what von Hippel calls "lead users" (von Hippel, 2005), in that many of them are technically sophisticated, and thus empowered users who are able to both innovate and effect technological change on a platform owned by a powerful producer (Apple). As von Hippel counts open source developers and mountain bike hobbyists-turned-manufacturers as examples of lead users, independent third party Cocoa developers, who do not work for Apple but often write iPhone or Macintosh apps for themselves, can be seen as innovating "users" in this light. Thus, this project will be a contribution to technology studies, user studies, as well as innovation studies.

Gender studies of technology have explored the effects of male domination of professional engineering disciplines and the cultural association of technology with masculinity, both at work and as hobbies (Cockburn & Ormrod, 1993; Cowan, 1983; Haring, 2003; Oldenziel, 1999; Oudshoorn, 1999; Wajcman, 1991, 2004). Recent studies have turned towards the male gendering of computer tinkering and programming (Ensmenger, 2009; Faulkner, 2000a, 2000b; Kleif & Faulkner, 2003; Lindsay, 2003). Although there are growing numbers of women in the computer engineering and programming professions, these fields

continue to be defined by masculine codes that value the pleasure in tinkering and hacking. Cocoa programmers, many of whom work for themselves or for very small companies, and say they are in it for the pleasure, tend to be overwhelmingly male. Thus this project will be also be a contribution to the study of homosocial technical worlds.

**Research Plan, Methods and Sources**

This project has two components, a historical component and an ethnographic one. The historical component will follow the social history of the NeXT developer community in the 1990s and its transformations in the wake of first the Apple purchase, and second, the release of the iPhone App Store. The purpose of this portion of the study is to explore the historical connection between the NeXT/Cocoa community and larger debates about the merits of dynamic object-oriented programming technology within the business and academic software community more widely, and how these debates helped to structure the subject position of being a NeXT/Cocoa programmer. This portion will be based primarily on oral history, consisting of semi-structured interviews in which a rough outline will be followed, but the interview will be flexible to allow for the exploration of topics arrived at *in situ*. Within the Cocoa community a number of longtime developers who have been on the platform since the NeXT days are well known. These include Andrew Stone, a developer of graphics applications, Ken Case, co-founder of Omni Group, one of the oldest independent Cocoa development companies, Wil Shipley, Case's former partner who founded Delicious Monster, one of the most well respected companies in the community, and Scott Anguish, former head of StepWise, a NeXTStep consulting firm. Other luminaries in the community include Aaron Hillegass and Bill Cheeseman, authors of popular Cocoa programming books, and Scott Stevenson, Daniel Jalkut, Brent Simmons, and Craig Hockeberry, developers of popular Mac and iPhone applications. Many of these men (the community is overwhelmingly composed of men) have widely followed blogs; indeed, blogs constitute one of the main "meeting spaces" of a community that is geographically distributed. I may also seek to interview Brad Cox, the creator of the Objective-C language which is used by Cocoa, who is a key proponent of object-oriented programming as a solution to the software crisis. Archival research of Cox's written work and that of Alan Kay, whose ideas about object-oriented programming underlie the Cocoa philosophy, may also be pursued as necessary.

The other component of the project is a multi-sited ethnographic study of a particular Cocoa development community located in Seattle. Seattle is the home of both Omni Group and Delicious Library, and the presence of these two well-known companies (or possibly, the aura of their founders) has attracted many younger Cocoa developers. (It remains an open question what effect the local presence of Microsoft has on this community, which sees Microsoft and its technology as the antithesis to the values of Apple and Cocoa.) The local community meets as a group called "XCoders" (a play on the name of Apple's development environment, XCode). I will seek to gain access to this informal group and attend the biweekly meetings as a participant observer. In addition, I will try to gain ethnographic access to the workplaces of members of XCoders, or other Cocoa developer companies in the area. This may involve volunteering my services as a software tester or programmer.

My former occupation as a software engineer in Apple's Cocoa technology division gives me a unique level of access to this community. I have built up a network of contacts beginning with personal friends and former coworkers at Apple. One of the most important

contacts is a former Apple Cocoa Evangelist, whose job involved maintaining a close relationship with the third party community. In addition, some members of the community are former Apple employees themselves. Although Apple's strict policy of secrecy prevents access to current Apple employees, my contacts there have been happy to point me to acquaintances outside the company. The fact that a large majority of Cocoa developers work either alone, with a single partner, or in small, entrepreneurial businesses makes gaining access significantly easier than gaining access to a large corporation. In addition, I have conducted previous research in Silicon Valley on two small iPhone software companies; in addition to incorporating this research into the dissertation, the relationships formed during this research provide another source of contacts.

**Tentative Outline of Chapters**

**Chapter Outline:**

1) Introduction

   This section will provide introduce NeXT and the NeXT/Cocoa developer community, and the centrality of this core group in promoting Apple's cultural values and object-oriented methodology among new iPhone developers. It will also present the theoretical frameworks of ideology and technological frames and discuss the related literature. Additional relevant literature on constructivist and gender technology studies, history of computing, and ethnography of computing will also be reviewed.

2) The beginning of the NeXT community in the late 1980s and early 1990s.

   This section will look at the original market of NeXT developers as contractors for Fortune 500s and in Rapid Application Development, their initial forays into early web services with WebObjects, and connections to Smalltalk programmers in the financial sector. This will be examined in the context of the movement within the software industry towards Object-Oriented Programming and Brad Cox's vision of "Software Industrial Revolution." Did the NeXT community share any connections with the Macintosh developer community? How large was it? Why did these programmers decide to develop for this platform? How did they feel about NeXT's technology versus its relative market obscurity?

3) Transitioning to "Indie."

   This section will examine how NeXT's purchase by Apple, the release of Mac OS X and the development of Internet infrastructure in the dot.com era transformed the community into independent applications developers for a consumer market. How did they negotiate a new identity as Macintosh developers? Were there any tensions between NeXT and Macintosh cultural and technical values, and how were these resolved? In what ways did the two sets of values combine and reinforce each other?

4) The Cocoa community on the Web

   This chapter will look at the role of blogs, twitter feeds, and mailing lists in constructing a virtual online space for like-minded Cocoa developers, how these

forums shape the discourse of the community, do boundary work, and construct positions of cultural and technical authority and expertise.

5) The Seattle XCoders

   This chapter will look at the dynamics of the Seattle XCoders developer group, and the role it plays in creating collegiality among competing local Cocoa entrepreneurs.

6) The iPhone Gold Rush

   This chapter looks at the cultural tensions surrounding the rapid influx of programmers from outside the Cocoa community and the introduction of stronger capitalist values associated with investor interest and cutthroat competition. This will be examined through a specific case study of an iPhone startup in the summer of 2008.

7) Ethnography of a Seattle based Cocoa company

   This chapter will examine how Cocoa community norms and values play out in the everyday engineering practice of an independent Seattle-based Cocoa company. Are there differences between developers producing for the iPhone versus the Macintosh? How are feature requests, bug reports, and complaints from users translated into design decisions, how is the user represented, and how are these issues negotiated in tradeoffs with engineering or marketing priorities? What is the nature of the relationship of the company or its individual developers with Apple? How might a positive relationship benefit the company, and how are tensions negotiated?

**Bibliography**

Agalianos, A., Whitty, G., & Noss, R. (2006). The Social Shaping of Logo. *Social Studies of Science*, *36*(2), 241-267.

Akera, A. (2001). Voluntarism and the Fruits of Collaboration: The IBM User Group, Share. *Technology and Culture*, *42*(4). Retrieved from http://www.jstor.org.proxy.library.cornell.edu/stable/c116380

Akera, A. (2007). *Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research*. Inside technology. Cambridge, Mass: MIT Press.

Akrich, M. (1992). The De-Scription of Technical Objects. In W. E. Bijker & J. Law

(Eds.), *Shaping Technology/Building Society: Studies in Sociotechnical Change*, Inside Technology (pp. 205-224). Cambridge, MA: MIT Press.

Bardini, T. (2000). *Bootstrapping : Douglas Engelbart, coevolution, and the origins of personal computing*. Stanford Calif.: Stanford University Press.

Bardini, T., & Horvath, A. T. (1995). The Social Construction of the Personal Computer User. *The Journal of Communication*, *45*(3), 40-66. doi:10.1111/j.1460-2466.1995.tb00743.x

Belk, R. W., & Tumbat, G. (2005). The Cult of Macintosh. *Consumption, Markets & Culture*, *8*(3), 205-217. doi:10.1080/10253860500160403

Bijker, W. (1995). *Of Bicycles, Bakelites, and Bulbs: Toward a Theory of Sociotechnical change*. Cambridge Mass.: MIT Press.

Bowker, G. C., & Star, S. L. (1999). *Sorting Things Out: Classification and Its Consequences*. Inside technology. Cambridge, Mass: MIT Press.

boyd, D. (2006). Friends, friendsters, and top 8: Writing community into being on social network sites. *First Monday*, *11*(12). Retrieved from http://www.firstmonday.org/issues/issue11_12/boyd/

Campbell-Kelly, M. (2003). *From airline reservations to Sonic the Hedgehog : a history of the software industry*. Cambridge Mass.: MIT Press.

Campbell-Kelly, M., & Aspray, W. (1996). *Computer: A History of the Information Machine*. The Sloan Technology Series (2nd ed., Vol. 1). Boulder, CO: Westview Press.

Campbell, H. A., & La Pastina, A. C. (2010). How the iPhone Became Divine: New Media, Religion and the Intertextual Circulation of Meaning. *New Media & Society*. doi:10.1177/1461444810362204

Ceruzzi, P. (1996). From scientific instrument to everyday appliance: The emergence of personal computers, 1970–77. *History and Technology: An International Journal*, *13*(1), 1.

Ceruzzi, P. E. (1998). *A History of Modern Computing*. History of computing; (2nd ed.). Cambridge, Mass. : MIT Press.

Cockburn, C., & Ormrod, S. (1993). *Gender and Technology in the Making*. London ; Thousand Oaks, Calif.: Sage.

Coleman, G. (2004). The Political Agnosticism of Free and Open Source Software and the Inadvertent Politics of Contrast. *Anthropological Quarterly*, *77*(3), 507-519.

Coleman, G. (2009). CODE IS SPEECH: Legal Tinkering, Expertise, and Protest among Free and Open Source Software Developers. *Cultural Anthropology*, *24*(3), 420.

Cooper, G., & Bowers, J. (1995). Representing the User: Notes on the Disciplinary Rhetoric of Human-Computer Interaction. *Cambridge Series On Human Computer Interaction*, (10), 48-66.

Cowan, R. (1983). *More work for mother : the ironies of household technology from the open hearth to the microwave*. New York: Basic Books.

Cowan, R. S. (1987). The Consumption Junction: A Proposal for Research Strategies in the Sociology of Technology. In W. Bijker, T. P. Hughes, & T. J. Pinch (Eds.),

*The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology* (pp. 261-280). Cambridge Mass.: MIT Press.

Cox, B. J. (1990a, October 1). There Is a Silver Bullet: A software industrial revolution based on reusable and interchangeable parts will alter the software universe. *BYTE*, Pg. 209.

Cox, B. J. (1990b). Planning the Software Industrial Revolution. *IEEE Software*, *7*(6), 25.

Eagleton, T. (1991). *Ideology : an introduction*. London ;;New York: Verso.

Edwards, P. N. (1996). *The Closed World: Computers and the Politics of Discourse in Cold War America*. Inside Technology. Cambridge, Mass.: MIT Press. Retrieved from http://www.h-net.org/review/hrev-a0a5x0-aa Materials specified: Book review (H-Net)http://www.h-net.org/review/hrev-a0a5x0-aa

Ensmenger, N. L. (2009). Making Programming Masculine. In T. Misa (Ed.), *Gender Codes: Women and Men in the Computing Professions*. Wiley.

Ensmenger, N. L. (2010). *The "Computer Boys" Take Over: Computers, Programmers, and the Politics of Technical Expertise*. Cambridge, MA: MIT Press.

Ensmenger, N. L., & Aspray, W. (2002). Software as Labor Process. In U. Hashagen, R. Keil-Slawik, & A. L. Norberg (Eds.), *History of Computing: Software Issues* (pp. 139-165). Berlin: Springer.

Faulkner, W. (2000a). The Power and the Pleasure? A Research Agenda for "Making Gender Stick" to Engineers. *Science, Technology, & Human Values*, *25*(1), 87-

119.

Faulkner, W. (2000b). Dualisms, Hierarchies and Gender in Engineering. *Social Studies of Science*, *30*(5), 759-792. doi:10.1177/030631200030005005

Freiberger, P., & Swaine, M. (2000). *Fire in the Valley: The Making of the Personal Computer* (2nd ed.). New York: McGraw-Hill.

Geertz, C. (1973a). Ideology As a Cultural System. In *The Interpretation of Cultures: Selected Essays* (pp. 193-233). New York: Basic Books.

Geertz, C. (1973b). *The Interpretation of Cultures: Selected Essays*. New York: Basic Books.

Gillespie, T. L. (2010). The Politics of 'Platforms'. *New Media & Society, Vol. 12, No. 3, 2010*. Retrieved from http://ssrn.com/paper=1601487

Greenberg, J. (2008). *From BetaMax to Blockbuster: Video Stores and the Invention of Movies on Video*. Cambridge  Mass.: The MIT Press.

Gusterson, H. (1996). *Nuclear Rites: A Weapons Laboratory at the End of the Cold War*. Berkeley: University of California Press.

Haigh, T. (2002). Software in the 1960s as concept, service, and product. *Annals of the History of Computing, IEEE*, *24*(1), 5-13. doi:10.1109/85.988574

Haigh, T. (2006). Remembering the Office of the Future: The Origins of Word Processing and Office Automation. *Annals of the History of Computing, IEEE*, *28*(4), 6-31.

Haigh, T. (2009). How Data Got its Base: Information Storage Software in the 1950s and 1960s. *IEEE Annals of the History of Computing*, *31*(4), 6-25.

Haring, K. (2003). The "Freer Men" of Ham Radio: How a Technical Hobby Provided Social and Spatial Distance. *Technology and Culture*, *44*(4), 734-761.

von Hippel, E. (2005). *Democratizing Innovation*. Cambridge  Mass.: MIT Press.

Hughes, T. P. (1987). The Evolution of Large Technological Systems. In W. Bijker, T. P. Hughes, & T. J. Pinch (Eds.), *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology* (pp. 449-482). Cambridge  Mass.: MIT Press.

Kahney, L. (2004). *The cult of Mac*. San Francisco: No Starch Press.

Kane, Y. I., & Catan, T. (2010, September 10). Apple Blinks in Apps Fight. *wsj.com*. Retrieved from http://online.wsj.com/article/SB10001424052748704644404575481471217581344.html

Kay, A. C. (1993). The early history of Smalltalk. In *The second ACM SIGPLAN conference on History of programming languages* (pp. 69-95). Cambridge, Massachusetts, United States: ACM. doi:10.1145/154766.155364

Keating, P., & Cambrosio, A. (2000). Biomedical Platforms. *Configurations*, *8*(3), 337-387.

Keating, P., & Cambrosio, A. (2003). *Biomedical platforms : realigning the normal and the pathological in late-twentieth-century medicine*. Cambridge  Mass.: MIT

Press.

Kellog, D. (2010, June 4). iPhone vs. Android. *Nielsen Wire*. Retrieved August 29, 2010, from http://blog.nielsen.com/nielsenwire/online_mobile/iphone-vs-android/

Kelty, C. (2008). *Two Bits: The Cultural Significance of Free Software*. Durham: Duke University Press.

Kleif, T., & Faulkner, W. (2003). "I'm No Athlete [but] I Can Make This Thing Dance!"--Men's Pleasures in Technology. *Science Technology Human Values*, *28*(2), 296-325. doi:10.1177/0162243902250908

Kline, R., & Pinch, T. (1996). Users as Agents of Technological Change: The Social Construction of the Automobile in the Rural United States. *Technology and Culture*, *37*(4), 763-795.

Laegran, A. S. (2003). Escape Vehicles? The Internet and the Automobile in a Local-Global Intersection. In N. Oudshoorn & T. J. Pinch (Eds.), *How Users Matter: The Co-Construction of Users and Technologies* (pp. 81-100). Cambridge Mass.: MIT Press.

Lessig, L. (2006). *Code and Other Laws of Cyberspace* (2nd ed.). New York: Basic Books.

Levy, S. (2001). *Hackers: Heroes of the Computer Revolution*. New York: Penguin Books.

Lindsay, C. (2003). From the Shadows: Users as Designers. Producers, Marketers, Distributors, and Technical Support. In N. Oudshoorn & T. J. Pinch

(Eds.), *How users matter: the co-construction of users and technologies* (pp. 29-50). Cambridge  Mass.: MIT Press.

Mahoney, M. S. (2002a). Software: The Self-Programming Machine. In A. Akera & F. Nebeker (Eds.), *From 0 to 1: An Authoritative History of Modern Computing* (pp. 91-100). New York, N.Y: Oxford University Press.

Mahoney, M. S. (2002b). Software as Science–Science as Software. In U. Hashagen, R. Keil-Slawik, & A. L. Norberg (Eds.), *History of Computing: Software Issues* (pp. 139-165). Berlin: Springer.

Mahoney, M. S. (2004). Finding a history for software engineering. *Annals of the History of Computing, IEEE*, *26*(1), 8-19. doi:10.1109/MAHC.2004.1278847

Mahoney, M. S. (2005). The histories of computing(s). *Interdisciplinary Science Reviews*, *30*(2), 119-135. doi:10.1179/030801805X25927

Mahoney, M. S. (2008). What Makes the History of Software Hard. *Annals of the History of Computing, IEEE*, *30*(3), 8-18. doi:10.1109/MAHC.2008.55

Malaby, T. (2009). *Making Virtual Worlds: Linden Lab and Second Life*. Ithaca: Cornell University Press.

Markoff, J. (2005). *What the dormouse said: how the sixties counterculture shaped the personal computer industry*. Viking.

van Meeteren, M. (2008, July 14). *Indie Fever: The genesis, culture and economy of a community of independent software developers on the Macintosh OS X platform* (Bachelor thesis, human geography). University of Amsterdam. Retrieved from

http://www.madebysofa.com/indiefever

Oldenziel, R. (1999). *Making technology masculine : men, women and modern machines in America, 1870-1945*. Amsterdam: Amsterdam University Press.

Oudshoorn, N. (1999). On Masculinities, Technologies, and Pain: The Testing of Male Contraceptives in the Clinic and the Media. *Science, Technology, & Human Values*, *24*(2), 265-289.

Oudshoorn, N., & Pinch, T. J. (2003). *How Users Matter: The Co-Construction of Users and Technologies*. Cambridge  Mass.: MIT Press.

Pinch, T. J., & Bijker, W. E. (1984). The Social Construction of Facts and Artefacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. *Social Studies of Science*, *14*(3), 399-441.

Turkle, S. (1995). *Life on the Screen: Identity in the Age of the Internet*. New York: Simon & Schuster.

Turner, F. (2006). *From Counterculture to Cyberculture: Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism*. Chicago: University of Chicago Press.

Wajcman, J. (1991). *Feminism Confronts Technology*. University Park  Pa.: Pennsylvania State University Press.

Wajcman, J. (2004). *TechnoFeminism*. Cambridge ; Malden  MA: Polity.

Woolgar, S. (1991). Configuring the User: The Case of Usability Trials. In J. Law (Ed.),

*A Sociology of Monsters: Essays on Power, Technology, and Domination*, Sociological review monograph (pp. 58-97). London; New York: Routledge.

Wyatt, S. (2003). Non-Users Also Matter: The Construction of Users and Non-Users of the Internet. In N. Oudshoorn & T. J. Pinch (Eds.), *How Users Matter: The Co-Construction of Users and Technologies* (pp. 67-79). Cambridge Mass.: MIT Press.