THREE ESSAYS ON DIGITAL EVOLUTION

A Dissertation Submitted

to the Temple University Graduate Board

In Partial Fulfillment

of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

By

Zhewei Zhang

May 2016

Examining Committee Members:

Youngjin Yoo (Advisory Chair), MIS Department, Temple University

Sunil Wattal, MIS Department, Temple University

Rob Kulathinal, Biology Department, Temple University

Nicholas Berente (External Reader), Department of MIS, University of Georgia

ABSTRACT

Digital products are rapidly shaping our world into a ubiquitous computing world. Because of its unique characteristics, digital artifacts are generative and highly evolving through the recombination of existing elements as well as by the invention of new elements. In this thesis, I first propose an evolutionary view to examine how digital artifacts can evolve over time based on the discussion of limitations of existing approaches. Then, inspired by contemporary evolutionary thinking and phylogenetics in biology, I develop a set of analytic approaches to classify different evolutionary patterns based on the evolutionary rate and evolutionary direction of the digital artifact. In three studies, I investigate digital evolution in specific contexts: web Mashup development, open source software development and digital design practice. I conclude my dissertation by highlighting the contributions of my work, and summarizing current limitations and opportunities for future research.

ACKNOWLEDGEMENTS

Completion of this doctoral dissertation was possible with the support of a number of individuals. I would like to express my sincere gratitude to all of them. First, I would like to thank Dr. Youngjin Yoo, my dissertation advisory chair, for his patient guidance and consistent encouragement throughout my research work. He is the funniest, smartest and most enthusiastic and energetic professor I know. He showed me the importance of keeping passion and keeping writing. I would also like to express my great appreciation to Dr. Rob Kulathinal, without whom I would probably never have gone on the journey of studying evolution. His expertise in natural science has significantly broadened my horizon and made this thesis possible. My sincere thanks also goes to Dr. Sunil Wattal, who provided me many suggestions and comments and often acted as gate-keeper to make sure my research is as rigorous as possible.

In addition to my committee members, I also give my special appreciation and thanks to several faculty members I have worked with as coauthors on different projects: Drs. Nicke Berente, Kalle Lyytinen, David Schuff, Susan Mudambi, and Habin Lee. Lessons learned from these projects also greatly helped my thesis writing.

I would like to thank the various faculty members in Temple's MIS department – Drs. Munir Mandviwalla and Paul Pavlou – for their help in research, teaching and coursework as well as the assistantship to support my school life. I am also very appreciative of the feedback on my dissertation essays provided to me by several colleagues: Sungyong Um, Aron Lindberg, Xinyu Li, and Jingbo Yu.

Last, but not least, I would like to express my deepest appreciation to my family for always supporting me. Words cannot express how grateful I am to my mother (Xiujun), father (Yuanzheng), my mother-in law (Lan), and father-in-law (Hongen), for all of their support. And of course, to my wife, Qingqing, thank you for your understanding, your patience, your confidence and your encouragement, and to my lovely daughter, Sandy, thank you for always making me smile no matter how tired I was.

I dedicate this dissertation to

my family, my wife, Qingqing, and my daughter, Sandy,

for their constant support and unconditional love.

I love you all dearly.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Digital technology is rapidly reshaping our world. As a result of transforming previously non-digital artifacts into digital form, a spectacular array of information is now digitally created, stored and consumed (Kallinikos 2007). The unique characteristics of digital artifacts make them generative and highly evolving through the recombination of existing elements as well as by the invention of new elements (Yoo et al. 2010a; Zittrain 2006). Yoo et al. (2010a) argue that the homogenization of data and re-programmability of digital technology – the two foundational properties of digital technology artifacts – have led to the emergence of layered modular architecture, which separates functionalities once belonging to the same design domain into different domains. The separation of functionalities in digital artifacts, combined with the pervasive availability of affordable digital technology, has enabled the design process of digital artifacts to be open and distributed, which further amplifies the generative nature of digital artifacts as evidenced in the case of the Internet.

Digital Evolution and Digital Artifact

For the purpose of this thesis, "digital evolution" refers to the phenomenon that digital artifacts are continuously and rapidly changing and developing without a clearly stable state like physical artifacts may have and therefore are creating all kinds of new variations from existing digital artifacts. "Digital artifacts" are mainly two types of artifacts: (a) man-made objects which can be created/stored/edited in digital format, such as software, or (b) human activities which are supported by digital tools and conducted in the digital space, such as automated digital design. Because changes in digital format can

be much easier to make, innovation to digital artifacts often comes through the participation of vastly distributed and sometimes even uncoordinated participants across different geographic locations or organizations. Boland et al. (2007) use the image of "wakes of innovations" to capture the ever-changing landscape as innovations from heterogeneous communities collide with one another. This produces highly volatile and dynamic patterns of innovations which drive continuous digital evolution. Thus, although digital artifacts are not always associated with large participant communities, in this essay I consider participant community to be an essential characteristic of digital artifacts and only focus on the digital evolution with community support.

An Evolutionary View of Digital Artifacts

In this thesis, I offer an evolutionary view to better understand how digital artifacts change over time. This view is supported by a computational approach to study the evolution of generative digital artifacts. In recent years, a large volume of digital trace data has become available because of technological advances. Human action can now be easily traced and recorded through logs, emails, videos and so forth. Especially for digital artifacts, every change can be logged by computer, such as in a revision log for software development, communication log for human designer or experimental result log for digital design. Those digital trace data provide the possibility of focusing on the process of social interaction and conducting micro-level computational inquiry, which leverages the data analytics supported by advanced computational analysis tools.

I propose the evolutionary view as a new approach of social inquiry for several reasons. First, when doing empirical studies, scholars usually rely on either a co-variance-based approach or a network-based approach to conduct quantitative studies.

These two approaches offer a rich set of evidence to explain the relationships among different factors. However, as Lazer et al. (2009) pointed out, these traditional analyses are generally based on "one-time" data points (even time-based analyses such as longitudinal analysis or dynamic network analysis are based on "repeated snapshots"), in comparison to continuous digital trace data, and are better to provide a macro-level picture of social interaction while lacking the ability to provide a comprehensive picture of complex social dynamics.

In general, the evolution of digital artifact is normally difficult to capture through these two approaches; the evolutionary process is often aggregated into several data points in order to fit into existing models. For example, on one hand, co-variance-based analysis, such as regression analysis, focuses on co-varying relationships between different attribute variables to establish their latent relationships and show how changes in some attributes may lead to certain outcomes in order to test a proposed hypothesis (Hair et al. 2006). How these relationships work is often conjectured and discussed through logical or theoretical reasoning during hypothesis development instead of in the actual analysis. On the other hand, a network-based approach (Wasserman et al. 1994) focuses on the structural relationships among different nodes, such as people or organizations, as the driving factor of certain outcomes. This approach often ends up with similar econometric analysis as the co-variance-based approach by testing the co-varying relationship between the network structural properties and outcome measurement. Both approaches therefore emphasize the state rather than the process.

Secondly, many qualitative researchers use a narrative approach to detail the evolutionary process. Often, one or more technology cases were studied longitudinally to

trace major technical and/or social events to examine how technology evolved (Anderson et al. 1990; Andersson et al. 2001; Tushman et al. 1998). However, due to the technical complexity of digital products, this approach often can only depict the higher level changes such as change of major components. A lot of nuanced changes taking place at lower levels, such as the source code level, are generally ignored, which only gives us an incomplete picture of the evolutionary process.

The key ideas of the proposed evolutionary view are that (a) digital artifacts will continuously change and never become stable as physical artifacts may, (b) the changes in digital artifacts take place in a temporal and sequential manner, and (c) there will be hidden patterns underlying the evolutionary process. Such a process pattern can be used as an alternative to using attributes or structural relationships to explain how a complex social world works. For example, how an open source project is developed and revised by various developers from the initial release to the current release and how its existing evolutionary process may lead to some future release.

Technology evolution has long been considered as the key driver of innovation. Prior scholars studying technology innovation, such as Tushman et al. (1986) and Anderson et al. (1990), consider technological discontinuities and dominant design as the primary pattern of technological change. Their works are primarily based on the changes of artifacts with physical format in which technology evolves through a long period of minor changes after the rise of a dominant design and is punctuated by technological breakthroughs which then lead to a new dominant design. Because digital artifacts are rapidly changing all the time, which significantly changed the basis of these prior researches, understanding the evolution of digital artifacts is even more important that

how the format from physical to digital may create new innovation patterns and whether the existing insight from physical artifact evolution still hold remains unclear.

Moreover, as our world is becoming more digitized every day, there is also an urgent practical need for policy makers to understand and predict the evolution of digital artifacts in order to be proactive about the potential effects brought by the digital artifact. For example, the Tor network was originally developed for the national security but later open sourced. Its subsequent use for criminal activities in the Dark Web should have been predicted based on the open source development of this project. Management teams in various technology companies also need to understand digital artifact evolution to help them position their products to compete with others as the evolutionary pattern is heavily affected by competition.

Aside from the evolutionary view, a computational approach inspired by techniques used in molecular biology is developed and used in my thesis. It includes two analytic techniques: organizational genetics and organizational phylogeny. The term "organizational" is used here to distinguish these genetics from their biological counterparts. The key to the organizational genetics approach is a cross-level analysis with a focus on complex interactions between the system and its underlying "genetic" elements. This approach helps us to understand the complexity of the phenotype (i.e., the distinguishing characteristic of the whole system) via the combination of its genotypes (i.e., the basic genetic building blocks of the system). Gaskin et al. (2014) proposed a way to code organizational routine based on several genetic elements to capture the details in design actions. The principle of this technique is to open the black box of a

previously sealed entity. Similarly, in my study, I open the digital artifact based on its unique factors and try to examine how different factors may affect its evolution.

Built upon the concepts of organizational genetics, organizational phylogeny further advances the analysis by using a specialized hierarchical clustering technique to build a phylogenetic tree of a digital artifact so that the evolutionary process can be observed. From the phylogenetic tree, the ancestral relationship and the branching history can be restored to help us better recognize evolutionary details. In the essays that follow, I discuss analytic details applied in each scenario.

Together, I propose this evolutionary view to systematically examine the changing nature of digital artifacts. It is not only a theoretical framework but also supported by a set of analytic analyses. It complements conventional analysis methods by focusing on the process rather than the state to provide new insights.

Epistemology of Evolution

Before moving forward, I believe it is necessary to first discuss in detail the different views on evolution in natural science and social science. In the Oxford English Dictionary (OED), the top two definitions of evolution[1] are: "the process by which different kinds of living organisms are thought to have developed and diversified from earlier forms during the history of the earth" and "the gradual development of something, especially from a simple to a more complex form." It is easy to see that the first definition mainly points to biological evolution, particularly Darwin's theory of evolution, which the OED further explains "helped Darwin toward a theory of gradual evolution over a

[1] http://www.oxforddictionaries.com/us/definition/american_english/evolution

long period by the natural selection of those varieties of an organism slightly better adapted to the environment and hence more likely to produce descendants." This definition, which is slightly different from the first, mainly refers to the use of evolution in explaining social changes. By comparing the two definitions, we see that the commonality between "biological" and "social" evolution is the process of gradual development. In other words, many people simply use the term "evolution" as a way to denote "change."

While the first definition seems to be the most commonly used, and many people many even consider "evolution" to refer strictly to biological evolution or Darwinism's evolution, the second definition is actually older than Darwin's evolution. In the early seventeenth century, the word was used to refer to the action of "unrolling" based on its Latin root *evolutio*. The more general use of the word evolution to refer to a process of development has been widely used since the eighteenth century. Even though Darwin's evolution is probably now the most widely used meaning, Darwin himself intentionally avoided using the word in *On the Origin of Species* in an effort to make his idea distinct from earlier research related to evolution. In fact, geologist Charles Lyell was the first scholar to use the term evolution in scientific writing. Thus, the word evolution was already being used before Darwin to describe the process of gradual development that follows a certain order, which generally implied an improvement. This is part of the second definition of evolution, going from a simple form to a more complex form.

However, Darwin's idea had nothing to do with improvement or progress. His concept of evolution is only about diversity, and how gradual changes can lead to new speciation together with the impact of natural selection. Such changes may or may not be

seen as improvements. Therefore, we can tell how natural science and social science use the word evolution differently: in modern biology, based on Darwin's theory of evolution, evolution predominantly concerns diversity as a result of gradual change, while in social science, both meanings have been used. The word is often used to refer to the gradual change process in social contexts and sometimes it implies progress to a better form or stage as a result of the gradual change. It is also used as an analogy to Darwin's evolution when scholars such as evolutionary economists try to apply the evolution mechanisms, such as natural selection, to explain social changes.

Evolutionary Thinking in Social Science

Evolutionary thinking has long been used by social scientist to explain social change, even before Darwin formulated his theory. French philosopher Auguste Comte's view of social evolution discussed three phases that society will undergo. Herbert Spencer also developed an evolutionary view in his metaphysics theory, which treats evolution as a progressive development process. Similarly, Karl Marx's evolutionary model of society also views society progressing in an ascending stages towards to communism. Those early views of social evolution consider that changes are "deterministic," differing from Darwin's view of change as "probabilistic" (Stzompka 1993). This line of evolutionary thinking, aligned with the second definition of evolution, has been implicitly or explicitly adopted by many later social scholars to describe progressive changes.

Many social scientists have also been heavily influenced by Darwinism. For example, Veblen (1898) and Marshall (1907) use Darwin's concept as a metaphor to analyze economic changes. However, this early evolutionary economics thinking did not

gain prominence over neoclassical economic theory. In more recent years, Darwinist evolutionary thinking has resurged in social science. Campbell (1965) proposed the variation and selective retention model and detailed the application of natural selection in biology to organization. His work is followed by Nelson and Winter's evolutionary theory (1982) that emphasizes the dynamic economic process instead of state in classical economics and provides an alternative to explain technological progress and economic growth. Organizational ecologists Hannan and Freeman (1977) take an evolutionary view in organizational studies in a somewhat different way by trying to answer the question "why are there so many kinds of organizations?" Organizational ecologists examine firms at population level to understand the variation and the selection process in how organizations are born and die.

In addition, a group of interdisciplinary scholars studying complexity science also try to link evolutionary systems and social systems. They see evolution in a similar way as social system as a complex adaptive system and argue for the importance of emergence and self-organization in evolution (Kauffman 1993, 1995; Solé and Goodwin 2000). Some (Schneider 2000) have even demonstrated by computer simulations that evolution can take place as a complex adaptive system. Unlike earlier social scholars that only tried to apply evolutionary thinking in social context, complexity scientist also work with biologist to help better understand evolution by using complexity theory. For example, molecular biologist Woese(2004) argues that evolution should not be considered as a "linear" system, rather it is involved with many non-linear dynamics across different levels from cells to organisms to individuals and to populations.

In this thesis, I followed the idea of biological evolution and use the word "evolution" to refer the process of gradual change that creates diversified digital artifacts. My work complements prior studies in social and technological evolution but approaches the social-technical changes from a somewhat different angle. Compared with evolutionary economics, my work focuses on the process of diversifying, not the process of economic growth. I specifically examine how different digital artifacts emerge. My work also differs from prior ecological works by admitting the different roles of mutation and selection in creating diversity. Population ecologists highlight the role of environmental selection as an explanation of various forms of organization at a macro level, whereas my work focuses on micro-level mutation as an explanation of diversity in digital artifacts.

Evolutionary Mechanisms

The modern field of biology generally considers the mechanism of evolution[2] as having four basic processes: mutation, gene migration, genetic drift, and natural selection. The first three processes are primary evolutionary mechanisms to create individual variations, while the last one is generally considered a mechanism to filter out variations that fit into a specific environment[3].

Mutation refers to the alteration of the nucleotide of the gene from one type to another. Alteration can come in different forms, such as deletion, insertion and replacement. This is often seen as similar to developmental changes in the social context.

[2] http://evolution.berkeley.edu/evolibrary/article/evo_14

[3] Natural selection is no longer considered to be the primary mechanism of evolution with the proposal of neutral theory in molecular biology.

Gene migration, also referred as migration or gene flow, is the variation brought through gene transfer from one population into another population. Likewise, we see that changes in organizations can sometimes be triggered by transfer from other regions or cultures, such as how Japanese management principles changed US company culture in the 1990s. Gene drift happens when there is a sudden, often random, change in population that causes changes in the frequencies of alleles, different forms of the same gene, in a population. Gene drift normally has higher impact when the population is smaller. For example, if there are only four white rhinoceroses in the world and the only one with a short ear gene dies, the whole population is not likely to have any short-eared rhinoceroses in the future. If a sudden disaster destroyed the majority of one population and only very few survived, the variation of future offspring will be heavily affected by those survivors; this is also called the founder effect. We also see similar cases in the social context, where a significant event such as political reform suddenly eliminates certain types of organizations. The last one, natural selection, concerns how individual organism may have different survive and reproduction rate because the its fitness to certain environment. This is because individual organisms can have different phenotypic characteristics, which can affect how well this individual would fit in certain environment. For example, how a company or product may be successful or failure in a certain market is a common topic in social studies.

Only natural selection has been widely considered when applying evolutionary thinking in social science. This is not surprising since it is easy to connect natural selection with market selection or social selection, and Darwinism has a long tradition of

influencing social studies. However, in this thesis, I will emphasize mutation as another important process underlying the evolution mechanism.

Mutation in Contemporary Evolutionary Biology

In addition to Darwinism, Mendelian-mutationism is an important piece in modern evolution view, but it has received less attention by social scholars when applying evolutionary views in studying social changes. Mutationism is an alternative view on evolution to Darwinism and it emphasizes on the role of mutation at genetic level, in addition to natural selection, on driving the evolution (Bowler 1992; Stoltzfus et al. 2014). So far, evolutionary social scholars, such as evolutionary economists or organizational ecologists, have relied heavily on the classical Darwinist view that focuses on macroevolution and has natural selection as the primary evolutionary mechanism. However, contemporary views on evolution have united Darwinism and Mendelian-mutationism to provide a comprehensive view of macroevolution and microevolution (Mayr 2001). By emphasizing the role of natural selection, Darwin's evolutionary theory focuses on macroevolution to understand changes taking place on geological time scales across different species and populations. In contrast, Mendelian genetics emphasizes change at the gene level that occurs within a species or population during a relatively short time period. Since the 1960s, the emergence of molecular biology has further advanced the microevolution thinking as researchers have studied the changes in molecules and how genotypic changes can then lead to phenotypic changes and therefore drive evolution.

The basic assumption of mutation in modern biology is that the mutation is random, which means the alteration is not driven by the organism's need. Though

mutation is considered random, it can be induced by mutagens. In molecular evolution, mutation can be detected through comparative analysis such as sequence alignment to identify differences among two gene sequences.

Mutation in Social Science

The random nature of mutation creates all sorts of variations for natural selection. However, such randomness contradicts traditional beliefs that social activities or human behavior are not random. For example, one of the most influential and fundamental economic principles comes from rational choice theory, which argues that human individuals are rational and make decisions based on their interests. Although many scholars argue there are no perfectly rational human beings, they still agree that individuals are at least partially rational and do not make random decisions. When I say I am going to emphasize the mutation aspect of modern evolutionary thinking, I am not going against this traditional thinking. However, by carefully examining the use of mutation in the biological and social context, I argue that there seem to be different understandings of mutation by natural scientists and social scientists, which discouraged the application of micro-level evolutionary thinking in social science.

I will first explain why mutation can still be the case in social science. The reason biological mutation is considered random by biologists is that the change is not driven by the organism's need. The organism itself cannot decide how its genes may mutate. This means that the change is intended and controlled by the organism. This intentionality is essential to the randomness of mutation. Understanding this, we can see that the idea of mutation can be applied in many scenarios in social science. Using digital evolution as an example, the modifications made to a software program by a human designer are

embedded with the designer's intention, not the software's self-intention. Thus by definition, this mutation is random to the software. Change in an organization is a similar case.

At this point, some may argue that software or an organization does not have its own intention and is not comparable to a human being's intention. Therefore, I will explain the second reason why changes in artifacts (especially digital artifacts) can also be seen as mutation in some cases. One key characteristic of digital artifacts is the participant community involved in the development process. Changes in digital artifacts are not made by one or two individuals; rather, they are very often contributed by a large number of people. When we consider the community, instead of individual, as the source of change, "there are a thousand hamlets in a thousand people's eyes" and there can be thousands of possibilities of mutation. To the artifact, the mutation can be seen as random. For example, for a project hosted on Github, there might be hundreds of forks and each fork may be revised slightly different, which made by different owners.

Mutation and Recombination

Another concept related to mutation in modern evolutionary thinking is recombination. Unlike mutation, which is not a commonly used term in social science, the term recombination has been used in natural science and social science quite often, although the uses are rather different. In biology, recombination is strictly used to refer the mixture of genes from two parents during meiosis in the sexual reproduction process. It is an important concept in genetics and considered as an important evolutionary mechanism by molecular evolutionists.

Recombination is a micro-level process through which gene migration and genetic drift are carried out. Therefore, recombination is different from mutation such that recombination is about changes through a switch of a complete set of homologous chromosomes, whereas mutation is about changes through alteration of individual genetic elements like DNA. However, in social science, the term recombination is often used in a similar way as the term mutation. For example, Arthur (2009) considers recombination to be the fundamental mechanism of technological evolution because every new technology is made by recombining existing technologies. However, this type of "social" recombination is more like the biological concept of mutation than the biological concept of recombination. First, the source of change is exogenous to the technology because the change is introduced by a human designer; second, recombination of different technologies actually alters part of an existing technology by inserting or replacing some components from other technology; this is comparable to mutation in biology instead of recombination. Therefore, even though Arthur uses the word recombination to refer to the mechanism of technological evolution, he implicitly applied the idea of mutation in his work and demonstrates that mutation is the force underlying the technology evolution. Similarly, Baldwin and Clark (2000) discussed several modular operators, such as splitting, substitution, excluding, augmenting, inversion, and porting, that can be used to change the modular structure of complex system. They also stated that such modular systems are "tolerant of uncertainty" and "welcome experimentation."

Following this reasoning, I therefore argue that both Arthur and Baldwin and Clark agree that "random" mutation is possible in technology innovation when the changes involve different participants with different interests and expertise. In my thesis,

the idea of mutation can be applied in social contexts, especially digital evolution. The terms mutation and recombination used in earlier social scientific studies are indeed the same as mutation in biology. However, in the rest of my thesis, I will continue use the term recombination as prior social science scholars have in order to eliminate confusion. Table 1 presents a summary of key terms used in the social science and biology fields.

Table 1. Comparison of Terms used in Biology and Social Science

Terms	Biology	Social Concept	Difference
Mutation	Unintended changes that does not necessarily benefit the organism, taking place at individual genetic element level.	Changes that are completely random.	Randomness in biology only means the change has nothing to do with organism's intention.
Recombination	Combing two sets of chromosomes from parents during the meiosis.	Mixture of different components or modules.	Recombination referred in social science is actually mutation in biology sense.
Evolution	A gradual process of new organism forms developed and diversified from earlier forms.	A gradual process of development and progress.	Biological evolution does not assume the evolution is always towards to better forms. It mainly concerns the diversity.
Natural Selection	Organisms with favorable variations will survive under environment pressure.	Market selection or consumer selection to find the better ones.	Natural selection in biology is not always directional to find the best fit but rather to eliminate the less fit. In fact, modern biology's neutral theory argue that natural selection is not the major force of evolution as most variations will have similar fitness that won't be eliminated by environment.

Summary

Overall, by applying the evolutionary view, I seek to answer the question "How do digital artifact variations emerge?" By concentrating on mutation as the key evolution mechanism, I analyze the evolutionary patterns of different digital artifact types in order to understand the emergence of variation.

In this thesis, I present three essays about digital evolution, focusing on different types of digital artifacts and different perspectives of digital evolution. In the first and second essays, I emphasize recombination and mutation as two major mechanisms of digital evolution, respectively. In the first essay, I develop a novel analytic approach that can be used to quantitatively measure the evolutionary process and propose two important properties, evolutionary direction and evolutionary rate, to evaluate different evolutionary patterns. I then use a data set composed of web API and web mashup data collected from programmableweb.com to demonstrate different patterns of evolution. By examining mashups belonging to different evolutionary patterns, I discuss various factors affecting evolutionary direction and evolutionary rate.

In my second essay, I examine digital evolution in the context of open source software (OSS) project development on a recently emerged social coding platform, Github.com, and attempt to investigate how the OSS is evolving as its source code is modified by different uncoordinated developers. Leveraging the evolution framework, I built in my first essay, I identify four types of evolutionary patterns for OSS and discuss how market competition can affect the way OSS evolves.

The third essay focuses on the evolution of digital artifacts using the case of autonomous electronic design. I first discuss the role of digital tools used in each design approach in the context of electronic design practice and then propose hypotheses about how two design approaches supported by different uses of digital tools can have different impacts on design performance. To test the hypotheses, I use field data collected from a large design firm to simulate how digital design activities and digitalized design activities can change in different patterns and lead to different levels of performance improvement over time.

CHAPTER 2 ESSAY ONE - DIGITAL INNOVATION AND ORGANIZATIONAL GENETICS: THE CASE OF WEB APIS AND MASHUPS

Introduction

The rapid miniaturization of computer and communication hardware, combined with increasing processing power, storage capacity, communication bandwidth and more effective power management, has made our world increasingly digital (Yoo 2012). Digital artifacts have several unique material characteristics that make them generative through the recombination of existing elements as well as by the invention of new elements (Yoo et al. 2010a). Due to their unique characteristics, digital artifacts often evolve beyond the original intent of the designers (Yoo 2012; Zittrain 2006). However, this generative nature of digital innovations has not been carefully studied in the literature. Generativity refers to an "overall capacity [of technology] to produce unprompted change driven by a large, varied, and uncoordinated audience" (Zittrain 2006). Similarly, Boland et al. (2007) use the imagery of "wakes of innovations" to capture the generative nature of digital innovations from heterogeneous communities colliding with one another.

Not all digital artifacts, however, are generative in the same way. For example, Apple's iOS is generative in a different way from Google's Android or Blackberry. The differences in the generativity of these digital artifacts have arguably influenced their fates in the marketplace. Despite its importance, generativity remains a vague theoretical concept that is difficult to use to differentiate different digital artifacts. Therefore, a better understanding of generativity and how it may differ from one digital artifact to another

will help us better understand the contemporary digital world and further help organizations to gain some insight on how to create and manage their digital artifacts.

In the extant literature, technology change has been seen from two broad perspectives. The first perspective, from those scholars who study technology innovation, is the technology lifecycle (Anderson et al. 1990; Benner et al. 2002; Tushman et al. 1998). In this perspective, technology change is seen as an evolutionary process following a punctuated equilibrium model (Tushman et al. 1986). The evolution of technology is understood mostly as improvements in performance. Therefore, the underlying assumption is that technology fundamentally does not change once a dominant design appears until the next disruption takes place. Another perspective comes from the study of social construction of technology (Bijker 1995; Callon et al. 1986; Hughes 1993; Pinch et al. 1984) and focuses on how institutional and cultural meanings of technology emerge in a society through social construction, especially how a complex web of actors and artifacts produce a stable meaning of technology over time. Such a perspective looks at how a singular technology comes into a being through diverse and ambiguous processes of social construction.

Although these perspectives come from different intellectual traditions, they both consider that technology (often physical ones) becomes stable once it has been successfully introduced in the market. To the contrary, digital artifacts do not become stable once they are created. Instead, many digital artifacts continue to evolve and propagate into different usage contexts. What we need, thus, is a conceptual model and an empirical tool to handle the generative evolution of digital artifacts.

To address this gap in the literature, I develop a computational approach to study the evolution of generative digital artifacts. Specifically, I explore how combinations of a finite set of digital technology elements at a lower level can produce seemingly infinite varieties at a higher level of product design hierarchy (Arthur 2009; Clark 1985) in the same way that the varieties I observe in nature derive from the combinations of a few genetic elements. Based on my analysis, I propose a theoretical framework to classify digital artifacts based on their evolutionary patterns. By doing so, I seek to answer the following research questions:

1. What are the dimensions of evolution of generative digital artifacts, particularly in the case of web mashups?
2. What are the factors influencing the evolution of digital artifacts, particularly in the case of web mashups?
3. How can I classify different forms of generativity of digital artifacts based on different evolutionary patterns?

To address these questions, I outline a computational method for an evolutionary study of digital artifacts. In this study, I focus on the evolution of a class of web services called web APIs that, when put together, enable a variety of web-based applications known as mashups (also referred to as Web 2.0). Web APIs are particularly well suited to my study as they are designed to attract heterogeneous third-party developers to re-purpose products and services and to transform them into new ones (Bush et al. 2010).

In the following sections, I first discuss the unique nature of digital technology and how it affects the generativity of digital artifacts, followed by more detailed description of web APIs. I then describe a two-level evolutionary model of web APIs and

mashups and present my empirical approach. Based on my empirical study, I finally propose a two-dimension framework to classify differences in generativity. I conclude the essay with future plans and the implications of my study.

Literature Review

Immateriality of Generative Digital Artifacts

Several unique material characteristics of digital technology make it generative through the recombination of existing elements as well as by the invention of new elements (Yoo et al. 2010a; Zittrain 2006). First, since all digital signals are represented with 0's and 1's (homogenization of data), different types of digital content can be easily combined to create new content. This leads to the separation of content and medium, which used to be tightly coupled for analog content (Yoo et al. 2010a). Second, the re-programmability of digital technology means that digitalized artifacts can now be flexibly re-programmed and re-purposed relatively easily through software that can endow new functions to the same hardware (Yoo et al. 2010a; Yoo et al. 2010b). This leads to the separation between form and function so that the same tool can perform, in principle, an unbounded set of different functions (like clocks, calculators, word processors, or web browsers) (Hair et al. 2006; Mayr 2001). Finally, as digital technologies that are necessary to produce digital innovations have become more affordable, the entry barrier has been lowered for independent entrepreneurs to participate in creative activities with digital technology (Kallinikos 2007; Yoo 2010). The universal diffusion of both the PC and the Internet have largely democratized innovation as users and entrepreneurs from

any part of the globe can participate in innovative activities, thus opening the floodgates to unbounded and generative digital innovations (Campbell 1965; Zittrain 2006).

These three characteristics of digital technology have become the foundation for making digital artifacts generative and highly evolving. Yoo et al. (2010a) argue that the homogenization of data and re-programmability of digital technology led to the emergence of layered modular architecture. Kallinikos et al. (2010) similarly argue that modularity and granularity of digital artifacts make them more generative. Furthermore, granularity and modularity are the results of these four characteristics of digital technology. The pervasive availability of affordable digital technology that enabled the design process of digital artifacts to be open and distributed amplify the generative nature of digital artifacts as evidenced in the case of the Internet.

Ever since the first discussion of technological generativity by Zittrain (2006), generativity has emerged as a key feature of digital artifacts. However, the concept of generativity is theoretical and empirically underdeveloped. The questions that remain unanswered include whether the generativity of all kinds of digital artifacts is the same, and if it is different, what drives the differences.

Scholars have studied the drivers and the nature of technology evolution and offered insights about generativity (Arthur 2009). Kauffman and Maceready (1995) and Arthur (2009) argue that technological evolution is like biological evolution which is essentially an adaptive search process across space constrained by environment. Many studies have explored how external sources, sometimes interacting with internal sources, promote technological evolution. Clark (1985), for example, examines how the interaction between design choice and customer preference affects technological

evolution. Similarly, Saviotti and his colleagues (1995) found that inter- and intra-technology competition had different impacts on technological evolution. These prior works suggest environment and competition as two potential forces that can drive the evolution of digital artifacts (Chesson et al. 1988). Building on these studies on technology evolution and digital artifacts, we draw on the work of contemporary evolutionary biology to study the generativity of digital artifacts in the context of mashups and Web APIs.

Technological Evolution and Innovation

How technology evolves is a key question in innovation studies and the new product development literature. Technological innovation is often categorized by a dichotomous typology (2015) such as discontinuous/continuous or radical/incremental. For example, technological discontinuities (Tushman et al. 1986) and dominant design (Anderson et al. 1990) are widely considered to be the primary pattern of technological change in which technology often evolves through a long period of minor changes after the rise of a dominant design, punctuated by technological breakthroughs which then lead to a new dominant design. By contrast, Duchesneau and his colleagues (1979) and Hage (1907) considered innovations with clear yet risky new technology to be radical innovation. Later, by focusing on emerging modular design practice, Henderson and Clark (2001) refined the concept of incremental versus radical innovation based on changes in core technologies and relationships among these core technologies and further identified two intermedia types of innovation. Overall, though from different

perspectives, technological evolution is roughly classified into major change and minor improvement.

Many studies have empirically tested determinants influencing innovation types, such as organization strategy (Wasserman et al. 1994), market orientation , and environmental dynamism (Bowler 1992). However, these prior studies mostly focused on the outcome rather than the process of technological change to classify technical innovation and have mostly relied on data from traditional industries such as manufacturing, where innovation generally follows a model that is closed (Woese 2004) and bounded by firms, whereas digital innovation is often distinguished by its open innovation model. Therefore, how such change may affect technological evolution and further help us identify different evolutionary patterns remains in question.

Web APIs and Mashups

Web APIs provide a software platform designed to support data communication on the Internet. It is an essential way of mobilizing and integrating various web-based applications and data to build a website or mobile applications and is able to glue different websites, web-based applications and mobile applications together. Firms like Google, Amazon, Facebook and Twitter are aggressively using web APIs to expand their presence on the web independent from their main website. For example, more than 60% of eBay listings come from its APIs. Also, Twitter gets 10 times more inbound traffic through its APIs than its website.

A mashup is an application that combines data and functionalities provided through web APIs to create new services. It can be either a web page or smartphone

application that normally includes more than one web API. According to Programmableweb.com, as of July 2014, there are over 6000 mashups in the format of web pages. These mashups cover a broad range of applications from mapping services to social networking to music streaming to serious enterprise services. Mashup sites are typically built by recombining existing web APIs along with a developer's own codes to create their own user interface and some additional functionalities and control. Given the number of available APIs and the pace at which new APIs are created, there seem to be almost an infinite number of different combinations to build new mashups.

Previous research suggests that building a vibrant third-party developer ecosystem is crucial in digital innovations (Boudreau 2010; Gawer 2011; Tiwana et al. 2010) and that APIs are becoming an indispensable strategic option to build such a vibrant ecosystem. As such, firms continue to introduce new web APIs, modify existing ones, and purge old ones. How does this affect the generativity of their ecosystem? Through a network analysis on mashups, Weiss and Gangadharan (2010) find that only a few web APIs form the critical core of many mashups in the Internet, following a power law distribution. However, they did not explore if there are different forms of generativity among mashups. In this paper, I look at how different web APIs are combined to give birth to mashups and explore possible patterns by which combinations of web APIs produce mashups. To do this, I employ an evolutionary lens to examine the basic design elements that constitute web APIs and how different combinations of those design elements affect the evolution of mashups.

Recombinant Innovation and Modularity

The variety of mashups available as a combination of Web APIs can be viewed as a classic example of recombinant innovation based on modularity (Arthur 2009). Hargadon (2002) underlines the recombinant nature of innovation as a process of dissembling and reassembling existing ideas, artifacts and people to create a new one. The variety of mashups present on the Internet demonstrates how web APIs, as extant artifacts, can be repeatedly combined to create new innovations at the mashup level.

Innovation as a process of recombination is accelerated by modularity that facilitates autonomous innovation within components and modular innovation across modules (Baldwin et al. 2000). Ethiraj and Levinthal (2004) highlight the importance of modularity in enabling innovation within complex system components, and recombination of the components is used to create new product at the system level. Therefore, innovations in web APIs can be understood simultaneously at the mashup level through the recombination of web APIs, and at the web API level that is evolving on its own. Hence, I will need a theoretical and empirical approach that allows innovations at both levels simultaneously as well as interactions between levels.

Organizational Genetics Approach

In this study, I propose a computational approach facilitated by advances in evolutionary genetics to analyze the evolution of Web APIs and mashups. Similar approaches have already been applied in studies on organizational routines. Gaskin et al. (2014) propose a genetics-based computational approach to analyze sociomaterial routines by coding organizational routines into sequences of activities consisting of sets

of basic elements. I build on this by coding Web APIs and mashups with basic elements, but then go beyond basic sequence analysis by taking advantage of advanced phylogenetic analysis to examine different evolutionary patterns of digital artifacts. I call this an organizational genetics approach ("organizational" here is used to differentiate it from biological genetics). It allows us to analyze the evolution of human-made objects such as digital artifacts or organizational routines with precision by using contemporary tools from biological genetics. The key to an organizational genetics approach is cross-level analysis with a focus on complex interactions between the system and its underlying "genetic" elements (Stoltzfus et al. 2014). Such an approach helps us to understand the complexity of the phenotype (the distinguishing characteristic of the whole system) via the combinations of its genotypes (the basic genetic building blocks of the system). In my case, the phenotype is the appearance and usage of those web APIs and mashups, while the genotype is their internal design logic.

Multi-Level Structures of Web API and Mashup: Analogy to Biology

I characterize Web APIs and mashups using a multi-level structure as shown in Figure 1. A mashup is a syndication of multiple Web APIs with their data and functionalities, combined with its own client-side logic and data representation to offer new services. Web APIs designed for similar purposes (e.g., Google Maps and Bing Maps) are often used together in many web mashups, thus creating Web API domains. A web API, in turn, can be further characterized by a set of common base design elements that define its function and performance.

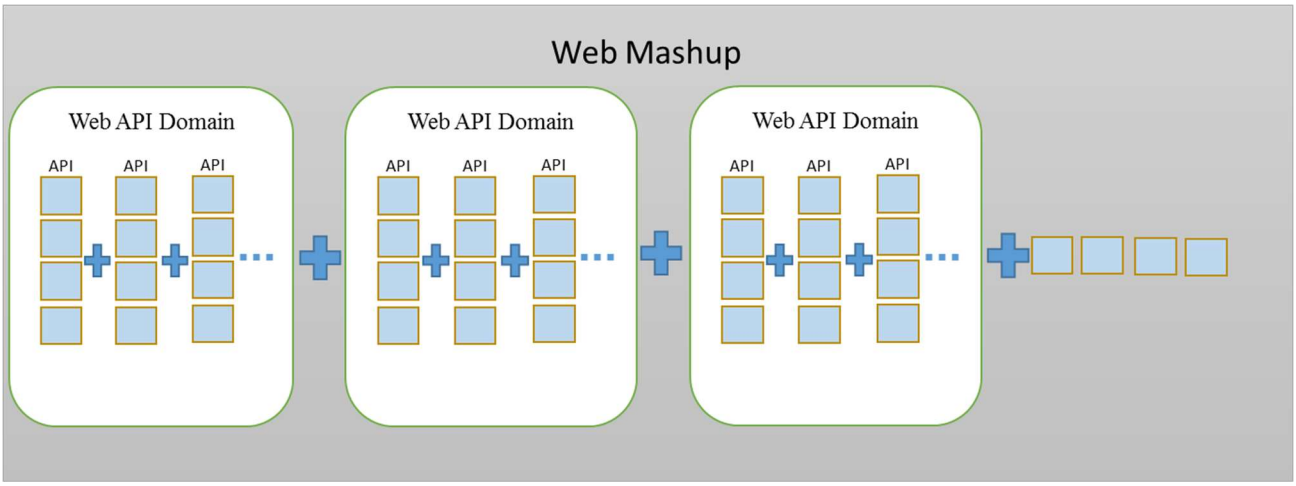

Figure 1. Multi-level Hierarchical Structure of Web API and Mashup.

As a digital artifact, web API needs to be represented with both social and technical aspects (Faulkner et al. 2010; Hair et al. 2006; Mayr 2001). In order to build a genetic model of web APIs and mashups, I need to identify a finite set of base design elements that can be used to characterize a Web API. Since there is no established way to determine the base design elements of Web APIs, I conducted an exploratory analysis using data I collected from Programmableweb.com to investigate possible candidates for base design elements. I then interviewed a panel of for web service experts from academia and industry via email, asking about a set of basic elements that can be used to classify web APIs and mashups. Based on the results, I characterize a web API with five base design elements with a fixed set of possible values (see Table 2).

Table 2. Base Elements and Examples

	Possible Values	Amazon eCommerce	OpenLayers	Mashup A
Service	New (news), Shp (shopping), Med (media), Son (social network), Map (mapping), S (advertising), Sec (search), Utl (utility), Com (communication)	Shp	Map	MapShp
Owner Type	Sin (single API owner), Mlt (multiple APIs owner)	Mlt	Sin	Sin
Openness	Opn (open), Cls (closed)	Cls	Opn	Cls
Function	Rtr (retrieve), Upl (upload), Upd (update), Rmv (remove)	RtrUplUpdRmv	Rtr	RtrUpl
Licensing	Fre (free), Pad (paid)	Fre	Fre	Pad
Subject Type	Api (web API), Msp (Mashup)	Api	Api	Msp

Service represents the domain that this API is designed for. According to Progammableweb.com, there are 54 different categories of service. Based on similarities of purpose, I grouped them into 9 groups of services, which cover almost all of the web APIs that are commonly used while keeping the computational complexity manageable. *Owner Type* indicates whether an owner has only one or multiple web APIs. *Openness* of an API reveals whether the source code is open or not. *Function* represents the software functions that the API provides, including "select," "add," "update," and "delete." Since Web APIs help users manipulate data from a remote server through a website, I use these as four possible values for service functions. Certain APIs have more than one function. For example, the Google Map API provides a retrieve function, while Twitter APIs provide both retrieve and update. My model permits such hybrid service functions. Finally, *Licensing* indicates if the API needs to be paid for to use it.

Using this coding scheme, I then code each API based on a sequence of values for these base elements in the order of Owner-Openness-Function-Licensing. Therefore, from Table 2, the sequence of Amazon API is [ApiMltClsRtrUplUpdRmvFrePad], OpenLayers is [ApiSinOpnRtrFre], and Mashup A is [MspSinClsRtrUplPad]. I further code each mashup as a concatenated sequence of its own code and the codes of APIs used. The order of APIs in a mashup is determined by the service domains. Therefore, complete coding for Mashup A that includes its own code as well as the web APIs it uses can be characterized as [MspSinClsRtrUplPadApiMltClsRtrUplUpdRmvFrePadApiSinOpnRtrFre]. This sequence represents the "DNA" of the mashup. A change of a mashup can occur through additions or deletions of APIs or through the mutation of an API as it might add new functions or change its licensing structure. In my analysis, such sequences have been further standardized so that they can be properly processed by the software.

Methodology

Data

I collected mashup and web API data from Programmableweb.com, an online database. Established in 2005, Programmableweb is considered the largest online repository tracking web APIs and Mashups, with more than 6300 mashups and 4600 web APIs with daily updates. It has been used as a data source in many academic publications and public press (Floyd et al. 2007; Yu et al. 2009). To assure accuracy, two research assistants with technical backgrounds coded each mashup independently. They were first trained with a coding schema. Three rounds of preliminary coding (250 mashups in each

round for a total of 750 mashups) were taken to achieve acceptable agreement. After the first round of coding had 56% agreement between the two coders, the coders and the first author had a discussion on disagreements before starting the second round, which ended with 70% agreement. After the second discussion, the inter-coder reliability between the two coders was 77% for the third round of coding. Finally, they were allowed to start working independently on the full data set.

I also checked the robustness of the arbitrariness of my sequence. Arbitrariness can be found within individual APIs (order of base components such as owner-openness-function vs. openness-owner-function) and between APIs (order of APIs such as advertising APIs-communication APIs-mapping APIs vs. mapping APIs-advertising APIs-communication APIs). The analytic approach I used in my paper, which will be discussed in detail below, relies on the sequence alignment, which produces a distance matrix that describes the pairwise difference between sequences. I performed a Mantel test on the distance matrices generated based on different arbitrary rules, and the results show the matrices are highly correlated (see examples in Appendix 2), suggesting that arbitrariness will not change the sequence alignment results. As long as relative positions of certain elements remain the same across different sequences, the difference between sequences remains.

Sequence Analysis

After the data collection was done, the raw data were converted into sequences. With the base design elements I defined earlier, I created sequences for mashups in two steps. I first define mashup as a sequence of web API domains (Figure 2) based on their

primary service. Therefore, I have 9 different web API domains. The absence of certain web API domains is replaced by gaps.

Figure 2. Sequence of Web API Domains.

In the earlier example of Mashup A from Table 2, only mapping (OpenLayer) and shopping (Amazon eCommerce) web APIs are used, therefore the sequence of web API domains for Mashup A looks like Figure 3.

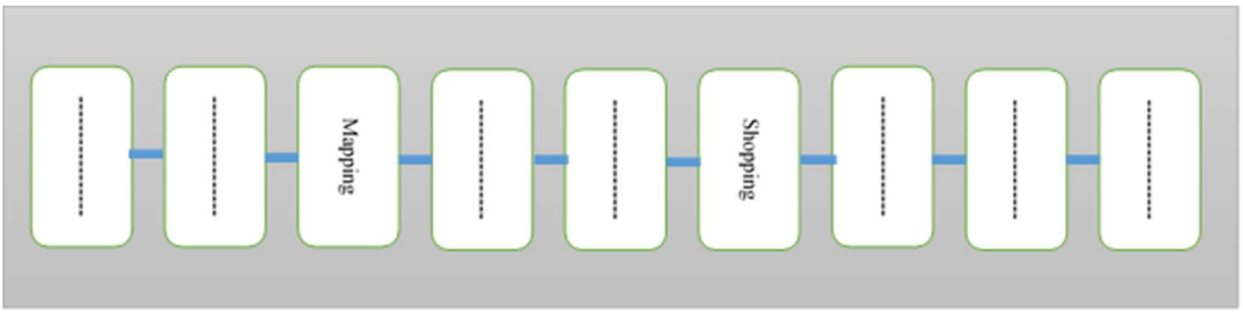

Figure 3. Example Sequence of Web API Domains for Mashup A.

In the second step, for each web API domain, I set a fixed number of APIs it can contain to ensure that the sequence lengths of the same web API domains in different mashups stay the same (Figure 4). The number of APIs that each domain can have is empirically determined by the maximum number of APIs that a mashup can have for that

specific domain. For example, the number of APIs in a shopping domain is four because in my dataset the maximum number of shopping APIs a mashup has is four. For a mashup created with a few web APIs in a certain domain, I again use gaps to note absence. In the earlier example, Mashup A only has one web API (Amazon eCommerce) in the domain for shopping, hence gaps serve as placeholders for the other four shopping APIs (see Figure 5). Figure 6 shows an example with the overall structure of mashup sequence. Rounded box represents web API domains, where grey ones are placeholders for absent domains in this example. Square represents individual web API, where red ones are placeholders.

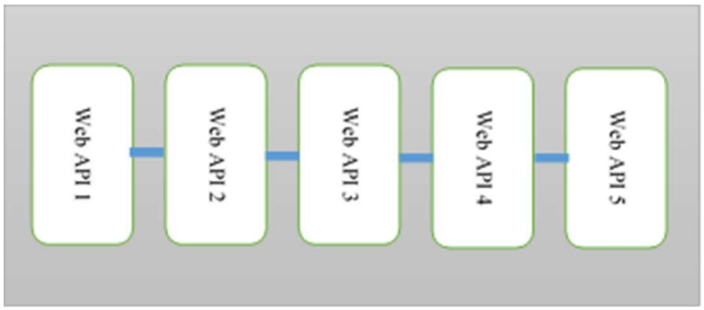

Figure 4. Sequence of Web APIs in a Domain.

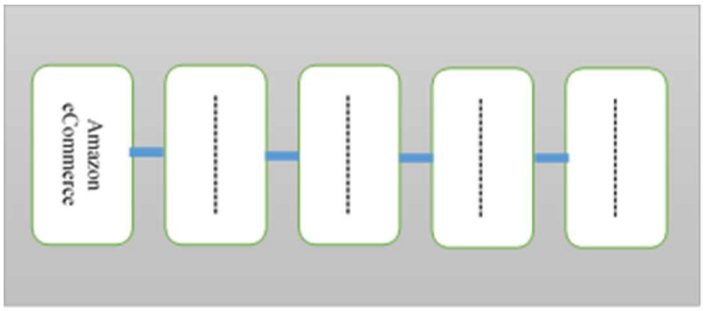

Figure 5. Example Sequence of Web APIs in Shopping Domain for Mashup A.

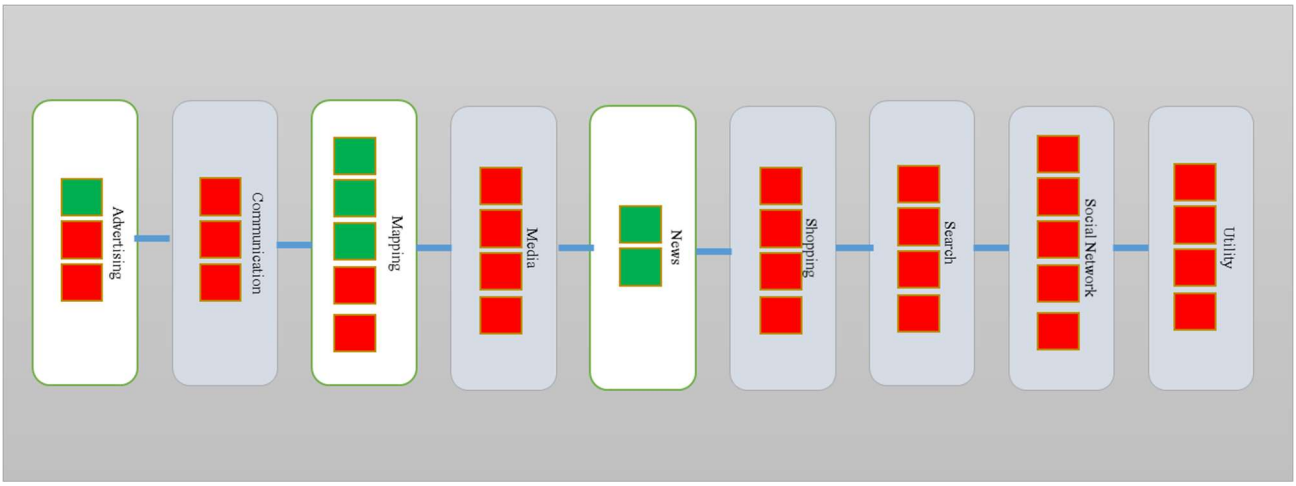

Figure 6. Overall Structure of Sequence for Mashup.

The last step is to build sequences for each individual web API based on the base elements I defined. Since I have already used the service context to create the level of web API domain, I use the other five elements (owner type, openness, service function, licensing, and subject type) to create the sequence, which naturally has the same length. Table 3 shows the sequences of three actual mashups.

Table 3. Examples of Complete Sequences

MspIndClsUtlRtrFre----------ApiIndClsRnDUnUPadApiIndClsRnDUnUPad---------- ----------
MspIndClsSeaRtrUpdFre---------- ----------ApiIndClsRtrFre----------
MspIndClsNewRtrFre---------- ApiIndClsRtrFre----------

I then completed sequence alignment using ClustalTXY, a sequence analysis software program specifically designed for the social sciences (Gaskin et al. 2014). ClustalTXY performs a pairwise alignment of the sequences and constructs a similarity

matrix that is converted into distance scores, which was used later for phylogenetic analysis.

Phylogenetic Analysis

Once I finished the sequence alignment, I then performed a phylogenetic analysis to explore evolutionary patterns of mashups. Alignment results were imported from ClustalTXY into TreeDyn and ATV provided via Phylogeny.fr (Dereeper et al. 2008). These are two software tools that are primarily used by evolutionary biologists to infer phylogenetic trees, estimate rates of evolution, and infer ancestral sequences. Using these tools, I was able to generate phylogenetic trees via a neighbor joining distance algorithm (Saitou et al. 1987). Neighbor joining is an algorithm, often used as baseline, to construct phylogenetic trees in biology. It is a button-up approach that starts at a random branch (mashup in my case) and then adds the nearest branch based on the distance matrix calculated during sequence alignment and iterates till all branches are added. Although other more recently developed algorithms are also available, I chose the neighbor joining method primarily for two reasons. First, it is much faster to process a large amount of sequences (in this case, thousands of mashups). Second, it can construct a tree correctly even when the evolution model is unknown or the correctness of the distance matrix is questionable (Mihaescu et al. 2009), which makes my result less sensitive to potential violations of biological assumptions.

In order to understand the result of phylogenetic analysis, I explain some basic notations used in phylogenetics using Figure 7. The root refers to the common ancestor of all taxa used, and mostly refers to an outgroup. In most cases, it is set with the oldest

taxa. The node represents a divergence event. Branches represent the evolutionary distance between two taxa, and a distance scale provides a reference to understand the magnitude of the difference between sequences. In this example, 0.1 means there is a 10% difference. Thus, branch length tells the actual difference between two sequences.

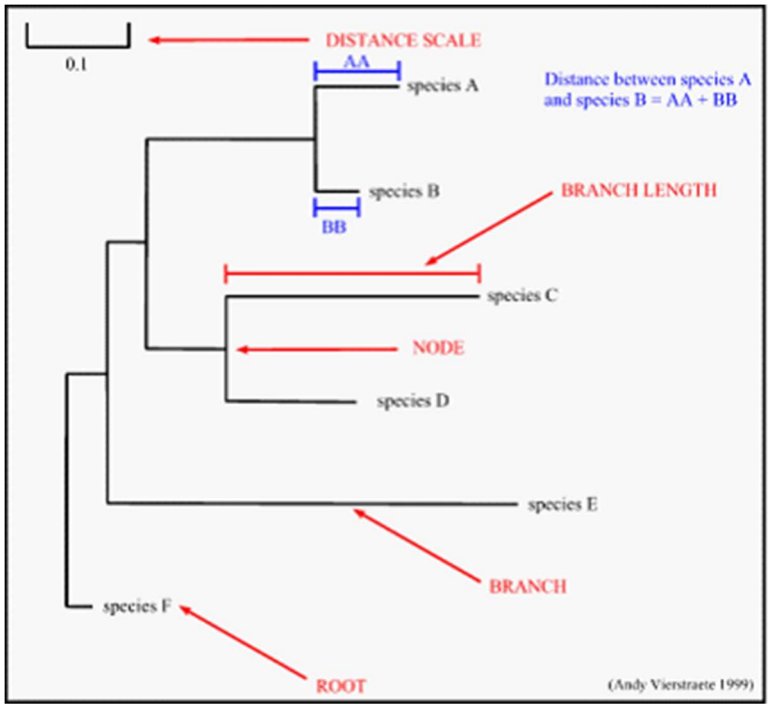

Figure 7. Example of a Phylogenetic Tree with Defined Notations.

An important property of tree shape is balance. Figure 8 below shows two extreme examples of trees with different balances. In general, tree balance measures how subgroups from tree root points are distributed. For example, tree A in Figure 8 is an example of a perfectly balanced tree in which every speciation point, including the root, has the same number of branches at left and at right. Tree B is an example of a perfectly imbalanced tree in which every speciation point, including the root, has only one taxa at left and all others at right. Tree balance is often used by evolutionary biologists to infer

macro-evolutionary phenomena regarding speciation. In my study, I follow the same approach to analyze the "speciation" of mashups so that I am able to understand how and why different mashups emerge.

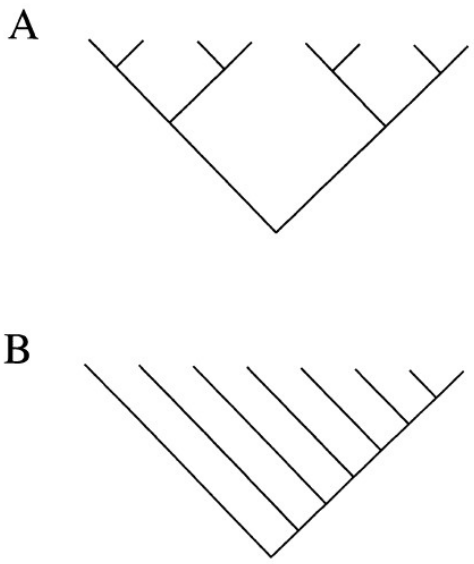

Figure 8. Example of Tree Balance.

The use of phylogenetic analysis has an advantage over traditional clustering analysis by taking time of change into consideration. Phylogenetic analysis allows researchers to infer temporal changes of the dissimilarities among individual objects. By setting the oldest object as the root, a phylogenetic tree showing the evolutionary pattern can be generated, which helps researchers to understand the process of how new members emerge.

Results

In this section, I present my results as follows: I first discuss the complete tree consisting of all mashups, which are grouped into several clades, or groups consisting of an ancestor-mashup and all its descendant-mashups, based on their evolutionary

relationship. Then I examine each individual group and investigate how each group differs from the others.

Phylogenetic Analysis of Web APIs and Mashup

A complete phylogenetic tree of all mashups can offer a high-level overview of how different mashups are related to each other along their evolutionary paths. Appendix 1A shows the phylogenetic tree without branch length information (different lengths here are just to provide a clearer view of the topology), and Appendix 1B includes branch lengths of the complete tree. The tree is rooted at del.icio.us dirctor, the oldest mashup[4] in my dataset. Based on the complete tree, I identify eight major clades that are significantly different from each other in terms of their genetic similarity. Three of the clades (1 at top and 2 at bottom) are close to the left and have mostly short branches, which suggests that mashups in these three clades did not evolve too much and are similar to each other within the same clades. The other five clades are more close to the right, which means they are more evolved. In addition to the relative left and right position of a clade in the complete tree, the relative overlapping or spread positions of branches within a clade also differs. Three clades have branches mostly overlapping with each other, meaning those mashups are similar to each other at the base element level, while mashups in other clades show more dissimilarities.

Trees generated in phylogenetic analysis can provide biologists with useful information to investigate speciation through two aspects: topology and branch length.

[4] Programmableweb stores the date a mashup was recorded by its database not the actual creation date. Therefore there are several "oldest" mashups in our data sample. Rooting with different mashups would change our tree slightly different but overall the same generally.

Topology is primarily concerned about how different taxa are related to each other, while branch length mainly focuses on evolutionary rates of the taxa. To understand the evolutionary pattern of each clade individually, I examine two important phylogenetic tree statistics: balance and branch length. I measure the balance of the phylogenetic tree for each clade by computing the standardized Colless Index[5] (Veblen 1898), a widely used measurement of imbalance in phylogeny. I also calculate the branch length within each clade. Both calculations were done using APTreeshap in R (Bortolussi et al. 2006). Below I report these statistics for each clade.

Results of Individual Clades

Clade 1 has the lowest balance index among the eight clades. A closer look at Clade 1 (Table 4) reveals the following key characteristics that make these mashups different from others. First, these mashups are primarily designed for using mapping services, particularly that provided by Google Maps API (used by more than 76.08% percent of mashups within this clade), with a secondary service focus. More than 68.11% of the mashups in this clade provide a wide range of secondary services in addition to mapping: the top secondary services are search (e.g., SkateSpots, a mashup to search nearby skate spots on map), shopping (e.g., slice_ny_pizza, which helps people to order pizza through nearby pizza restaurants displayed on the map), and utility (e.g., Gruvr, which helps to track live events around a given location). Therefore, Clade 1 is a Google

[5] I computed the Colless Index according to the Yule model as well as the PDA model. Both models gave us similar results in terms of relative difference among the different clades. Therefore I only reported results based on PDA model.

Map ecosystem. It has the lowest balance index of 1.6, suggesting a well-balanced ecosystem that yields a diversified outcome of innovation.

Table 4. Key Characteristics of Clade 1

Key Descriptions		Top Secondary Service			Top APIs used		
Balance Index	1.6	Search	50	16.61%	Google Maps	229	76.08%
Total Number of Mashups	301	Shopping	45	14.95%	GeoNames	40	13.29%
Average Age	1052	Mapping	30	9.97%	Twitter	33	10.96%
Primary Service	Mapping	Utility	24	7.97%	Flickr	30	9.97%
Evolutionary rate	0.064	Media	22	7.31%	Facebook	24	7.97%
		News	14	4.65%	YouTube	16	5.32%
		Social Network	13	4.32%	Microsoft Bing Maps	16	5.32%
		Communication	6	1.99%	Panoramio	11	3.65%
		Media	1	0.33%	Google AdSense	10	3.32%
		Grand Total	205	68.11%	Yahoo Maps	8	2.66%

Mashups in Clade 2 (Table 5) are predominantly designed for search services. Unlike Clade 1, in which a majority of mashups also provide secondary services, only 37% of the mashups in Clade 2 provide a secondary service, which implies a relatively low diversity in terms of service focus. However, the search services provided in Clade 2 are actually enabled by various APIs with different focuses. I found no dominating API in this clade. The most used API is still Google Maps, but it was only adapted by about 13.51% mashups, followed by Amazon eCommerce at 9.09% and Flickr at 4.91%. For exampke, Musipedia was a mashup aimed to become the music encyclopedia by providing ability to search web by using music/melody as input. Another example is

PriceNoia which allow shopper to search and compare products sold at 6 different Amazon international stores. The clade's low balance index of 2.41 suggests a rather diversified evolutionary direction achieved by combining different APIs.

Table 5. Key Characteristics of Clade 2

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	2.41	Utility	45	11.06%	Google Maps	55	13.51%
Total Number of Mashups	407	Shopping	40	9.83%	Amazon eCommerce	37	9.09%
Average Age	1283	Media	28	6.88%	Flickr	20	4.91%
Primary Service	Search	Mapping	18	4.42%	Google Search	19	4.67%
Evolutionary rate	0.05	Social Network	10	2.46%	Facebook	18	4.42%
		Search	6	1.47%	Google Ajax Search	14	3.44%
		Media	3	0.74%	eBay	12	2.95%
		News	1	0.25%	Bing	11	2.70%
		Grand Total	151	37.10%	del.icio.us	10	2.46%
				0	Twitter	9	2.21%

Clade 3 (Table 6) shares similar characteristics with Clade 2, although its primary service is utility. Only 22.2% of the mashups in Clade 3 offer a secondary service, which is the second lowest percentage among the 8 clades, suggesting the mashups in this clade are designed to offer similar solutions to users. The balance index of 2.92 is higher than in other clades, suggesting an unbalanced evolution tree, which means the evolution of mashups in this clade tend to be in a similar direction. The evolutionary rate of 0.016 is also low, meaning that the generic changes of mashups are small. Example mashups in

this clade include busAlert.me, which is a website which can track buses and send notices to users.

Table 6. Key Characteristics of Clade 3

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	2.92	Utility	61	6.33%	Google Maps	155	16.08%
Total Number of Mashups	964	Mapping	44	4.56%	Twilio	101	10.48%
Average Age	1444	Shopping	29	3.01%	Twitter	84	8.71%
Primary Service	Utility	Search	25	2.59%	YouTube	63	6.54%
Evolutionary rate	0.016	Communication	15	1.56%	Flickr	56	5.81%
		Social Network	16	1.66%	Facebook	54	5.60%
		Media	14	1.45%	Twilio SMS	50	5.19%
		News	9	0.93%	Box.net	48	4.98%
		Advertising	1	0.10%	Google Homepage	38	3.94%
		Grand Total	214	22.20%	Amazon eCommerce	36	3.73%

Clade 4 (Table 7) is similar to Clades 2 and 3 in terms of relatively large balance index (2.39) and small evolutionary rate (0.038). The primary purpose of the mashups in Clade 4 is to provide media-related services. Therefore, it is not surprising to see that YouTube (23.36%), Flickr (19.74%) and Last.fm (14.80%) are the most used APIs in this clade. SnapTweet is an example of a mashup in this clade, combining the Flickr and Twitter APIs to provide an easy way to share Flickr photos on Twitter.

Table 7. Key Characteristics of Clade 4

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	2.39	Utility	53	17.43%	YouTube	71	23.36%

Table 7. continued							
Total Number of Mashups	304	Search	22	7.24%	Flickr	60	19.74%
Average Age	1207	Social Network	20	6.58%	Last.fm	45	14.80%
Primary Service	Media	Mapping	9	2.96%	Twitter	27	8.88%
Evolutionary rate	0.038	News	6	1.97%	Google Maps	24	7.89%
		Communication	3	0.99%	Facebook	20	6.58%
		Shopping	3	0.99%	Amazon eCommerce	18	5.92%
		Social Networking	1	0.33%	Twilio	14	4.61%
		Grand Total	117	38.49%	MusicBrainz	9	2.96%
				0	ImageLoop	9	2.96%

Clade 5 (Table 8) differs from the clades described above in that there is no dominating service focus, which means that mashups in this clade are designed for many different purposes. The balance index of 1.93 suggests the phylogenetic tree is much more balanced than in the other clades. In other words, the mashups of Clade 5 tend to evolve in different paths. However, the APIs used by these mashups are still quite similar compared with other clades. Example mashups include SwingVine, a mashup to help users discover music, movies and other entertainment through social recommendations, which combines Google Contacts, Windows Live Contacts, Yahoo Address Book and YouTube APIs.

Table 8. Key Characteristics of Clade 5

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	1.93	Utility	64	22.61%	Twitter	104	36.75%
Total Number of Mashups	283	Search	9	3.18%	Twilio	59	20.85%

Table 8. continued							
Average Age	1456	Media	9	3.18%	Google Maps	45	15.90%
Primary Service	n/a	Mapping	6	2.12%	Facebook	37	13.07%
Evolutionary rate	0.28	Shopping	5	1.77%	Twilio SMS	29	10.25%
		Social Network	5	1.77%	Amazon eCommerce	14	4.95%
		Communication	4	1.41%	YouTube	13	4.59%
		News	4	1.41%	Flickr	12	4.24%
		Grand Total	106	37.46%	Geocoder	12	4.24%
				0	Foursquare	9	3.18%

The primary service provided by mashups in Clade 6 (Table 9) is to facilitate shopping experiences. The most common used APIs are Amazon eCommerce and eBay, which are the two most popular ecommerce companies. The balance index and evolutionary rate are similar to those of Clades 2, 3 and 4. Therefore, the mashups in Clade 6 are also evolving at a slow pace without too much diversity. For example, Savings.com uses 8coupons and Groupon APIs to retrieve local deal information from both sites and provides a one-stop service to find local deals.

Table 9. Key Characteristics of Clade 6

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	2.48	Utility	25	14.45%	Amazon eCommerce	49	28.32%
Total Number of Mashups	173	Search	21	12.14%	eBay	35	20.23%
Average Age	1456	Mapping	10	5.78%	Google Maps	26	15.03%
Primary Service	Shopping	Media	3	1.73%	Shopping.com	21	12.14%
Evolutionary rate	0.011	Social Network	1	0.58%	YouTube	8	4.62%

Table 9. continued							
		News	1	0.58%	Twitter	8	4.62%
		Utility	1	0.58%	Shopzilla	7	4.05%
		Communication	1	0.58%	Facebook	7	4.05%
		Shopping	1	0.58%	Zazzle	6	3.47%
		Grand Total	64	36.99%	Amazon EC2	5	2.89%

Clade 7 (Table 10) is probably the most unique clade in my data set. At first glance, Clade 7 seems look like Clade 1. For example, it has the same primary service in mapping, with Google Maps API used by more than 85.22% of its mashups. In addition, more than two-thirds of mashups in Clade 7 offer a secondary service along with mapping service. However, the major difference from Clade 1 comes from the details of the secondary services. Though 153 out of 230 mashups offer secondary services, all of them are offering the same type of service: utility. Some examples include Real Time Satellite Tracking, which allows users to track any known satellite on Google Maps, and NYC Bike Maps, which provides comprehensive biking routes and related information on a New York map. As a result of such concentrated service focuses, the balance index of Clade 7 is the highest among all clades, 4.5, which means the mashups belonging to this clade are evolving and emerging in a similar direction.

Table 10. Key Characteristics of Clade 7

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	4.5	Utility	153	66.52%	Google Maps	196	85.22%
Total Number of Mashups	230	Grand Total	153	66.52%	Flickr	13	5.65%

Table 10. continued							
Average Age	995				Microsoft Bing Maps	13	5.65%
Primary Service	Mapping				Amazon eCommerce	13	5.65%
Evolutionary rate	0.0009				YouTube	10	4.35%
					Twitter	9	3.91%
					Facebook	8	3.48%
					Yahoo Maps	6	2.61%
					Google Ajax Search	6	2.61%
					Panoramio	4	1.74%

Like Clade 5, Clade 8 (Table 11) also does not have a dominating primary service. Moreover, not many mashups offer secondary services as well; its rate of 18.65% is the lowest figure among all the clades. That is to say that mashups in this clade are likely to be designed with a simple purpose. Unlike Clade 5, where the usage rates of the most commonly used APIs are close to each with the highest percentage of 37%, Google Maps API is the most used API in Clade 8, and the usage is nearly 8 times more than that of the second most used API from Twilio. Both balance index (3.15) and evolutionary rate (0.147) are larger than most of the other clades. In other words, despite the simple service purpose, the mashups in this clade are changing nearly 14.7% of their genes on average with similar direction. One mashup example in this clade is Incident1, which displays police, fire and other emergency incidents on map.

Table 11. Key Characteristics of Clade 8

Key Numbers		Top Secondary Service			Top APIs used		
Balance Index	3.15	Utility	16	8.29%	Google Maps	106	54.92%
Total Number of Mashups	193	Social Network	5	2.59%	Twilio	14	7.25%
Average Age	1223	Shopping	4	2.07%	Flickr	11	5.70%
Primary Service	n/a	Media	4	2.07%	Twitter	11	5.70%
Evolutionary rate	0.147	Communication	3	1.55%	Yahoo Geocoding	11	5.70%
		Mapping	2	1.04%	Microsoft Bing Maps	9	4.66%
		Advertising	1	0.52%	YouTube	9	4.66%
		News	1	0.52%	Twilio SMS	8	4.15%
		Grand Total	36	18.65%	Facebook	7	3.63%
					Yahoo Maps	7	3.63%

Discussion

Scholars have considered the direction of evolution and rate of technological change as two important aspects of technological evolution in order to understand where and how technology evolves over time. With the help of phylogenetic analysis, I now can clearly see how mashups evolve differently within different clades with regard to those two key aspects. I also need to understand why those digital artifacts may evolve differently. I elucidate potential reasons based on my results and existing theories below.

Understanding the Direction of Evolution of Digital Artifacts

What drives the direction of evolution has been widely studied in biology, particularly in ecology studies on speciation. In recent decades, ecologists have started to leverage phylogenetic techniques to understand how different forces may drive the

evolution of ecological traits. In particular, by analyzing the shape of the phylogenetic tree of a species, ecologists can identify the dominant evolutionary force underlying the tree shape.

Webb et al. (2002) and Davies et al. (2012) focused on competition and environment as two major forces and how the switch between two forces may shift the tree shape. They used tree balance as the key measurement of tree shape to infer the dominant evolutionary pattern. Their findings suggested that tree balance/imbalance is largely the result of the relative strength of the competition and environment that drive evolution. When the environment is stable and competition becomes the primary evolutionary force, the tree will be more balanced because competition restricts the coexistence of species with similar traits (Grant 1972; Jonathan Davies et al. 2007). As species with similar traits exist in the same environment, they inevitably compete for similar resources. Losing this competition usually leads to evolution toward to different trait path to avoid competition, therefore exhibiting diversified evolutionary patterns. Thus it is more likely to have multiple branches at each splitting node (a gene), resulting in a balanced phylogenetic tree. In contrast, when the environment is highly volatile and fit to the environment becomes more critical for survival, species would be expected to share more similar traits (Webb 2000; Webb et al. 2002), which means the evolutionary pattern would be more uni-directional than former situation. Environmental volatility essentially prohibits other evolution paths that do not fit, leaving surviving species more like each other.

I analyze what may cause the evolutionary direction to be similar or dissimilar within a clade. I compare Clades 3, 7, and 8 to Clades 1, 2, 3, 4 and 5 (particularly

between 7 and 1 & 5, as differences among 2, 3, 4, 6, 8 are less). I did not find strong distinctions between the two groups of clades by looking at their genetic differences. Thus I suspect the difference may come from sources other than internal generic sources. As I argued earlier, in biology, the degree of tree balance reflects the dominance of competition and environment as two evolutionary driving forces. The higher the degree of tree imbalance, the more strongly the environment affects the evolution and vice versa. After reviewing my clades, I believe such rationale can also be applied in my case.

In the management literature, organizational ecologists (Hannan et al. 1977) had a similar idea in their niche theory that distinguished between generalist organizations and specialist organizations. They argued that generalism may not be the best choice in an uncertain environment. In my case, Clade 7 is much more influenced by the environment than Clade 1. The trace of environmental influence can be found in the observation that mashups in Clade 7 are primarily designed for providing mapping and utility services. Unlike mashups in other clades which tend to have different secondary services in addition to their primary service, mashups in Clade 7 only have one type of secondary service (utility), and 66% of the mashups did have secondary service. Furthermore, Google Map API was the dominating APIs, with more than 85% mashups utilizing it. Overall, Clade 7 is a highly specialized group of mashups that serve a niche market. In contrast, though a majority (68%) of mashups in Clade 1 also had a secondary service, they were much more diversified than in Clade 7. No secondary service accounted for more than 17%, compared with 66% in Clade 7. Based on this discussion, I propose:

PROPOSITION 1: When environmental constraint overwhelms competition pressure, digital artifacts will evolve toward the same direction, but when competition pressure

overwhelms environmental constraint, digital artifacts will evolve toward diverse directions.

Understanding the Evolutionary Rate of Digital Artifacts

Branch length of a phylogenetic tree is closely tied with evolutionary rate. As I discussed earlier, the branch length basically represents the genetic distance between two species, or mashups in my case. In biology, a phylogenetic tree is normally constructed with the basic assumption that substitution of gene components takes place at a constant rate, therefore the branch length aligns with evolutionary rate. Though the assumption of a constant substitution rate is often unrealistic, particularly during a long period of evolution, in my case, the whole evolution period is less than eight years. As such, I believe the substitution rate should hold the same across different clades (or at least change in a similar way). Therefore, the average branch length of mashups in each clade aligns with the evolutionary rate of each clade, and I then can use the branch length as the proxy for evolutionary rate.

Table 12 summarizes the tree balance and average branch length for each clade, ranked by the balance index. Clade 7 shows the smallest number of average branch length. Such small number presented the fact that very little evolution (less than 0.1%) taking place in this clade since the creation of first mashup. The most dramatic evolution happened in Clade 5, which had an average 28% change. In other words, on average each year the new mashups in this clade had 28% different genes compared with earlier mashups.

Table 12. Summary of Balance Index and Average Length for Each Clade

Clade	Balance Index	Branch Length
1	1.6	0.064
5	1.93	0.28
4	2.39	0.038
2	2.41	0.05
6	2.48	0.011
3	2.92	0.016
8	3.15	0.147
7	4.5	0.0009

In my sample, there are two clades classified as fast evolving clades: 5 and 8. The common characteristics between these two, which cannot be found in any other clades, is that there is no primary service context. In other words, the mashups in these two clades are diversified to serve different purposes. Since my analysis method generates clades of mashups based on the alignment of their genes, the reason that these mashups are still included in the same clade is the similar APIs used and the similar way APIs used. Therefore, even though these mashups are made up of similar APIs in similar ways, they evolve much faster than other clades because of the diversity of the designers' intention. I summarize my observation in the following proposition:

PROPOSITION 2: A diversified design purpose will lead to a faster evolutionary rate and vice versa.

Toward a Theoretical Framework of Generativity of Digital Artifacts

By combining analysis on topology and branch length, I get a better understanding of the evolution of a digital product. Generativity has been considered the key feature that differentiates new digital artifacts from physical artifacts. The balance index and evolutionary rate I discussed can be used as two dimensions to distinguish different types of generativity and provide a quantitative measurement of generativity. Based on my articulation of each clade, I proposed the following two-dimension framework of generativity and tried to explain the relationship between evolutionary pattern and clade characteristics.

In this framework, I propose evolutionary directionality and evolutionary rate as two dimensions to classify different types of generativity. I first standardize my data and then plot my clades in two-dimensional space with the X-axis as evolutionary rate and the Y-axis as evolutionary direction. Thus I tried to identify four major different types of generativity with different evolutionary patterns as shown in Figure 9 below. I summarize the four different patterns in Table 13.

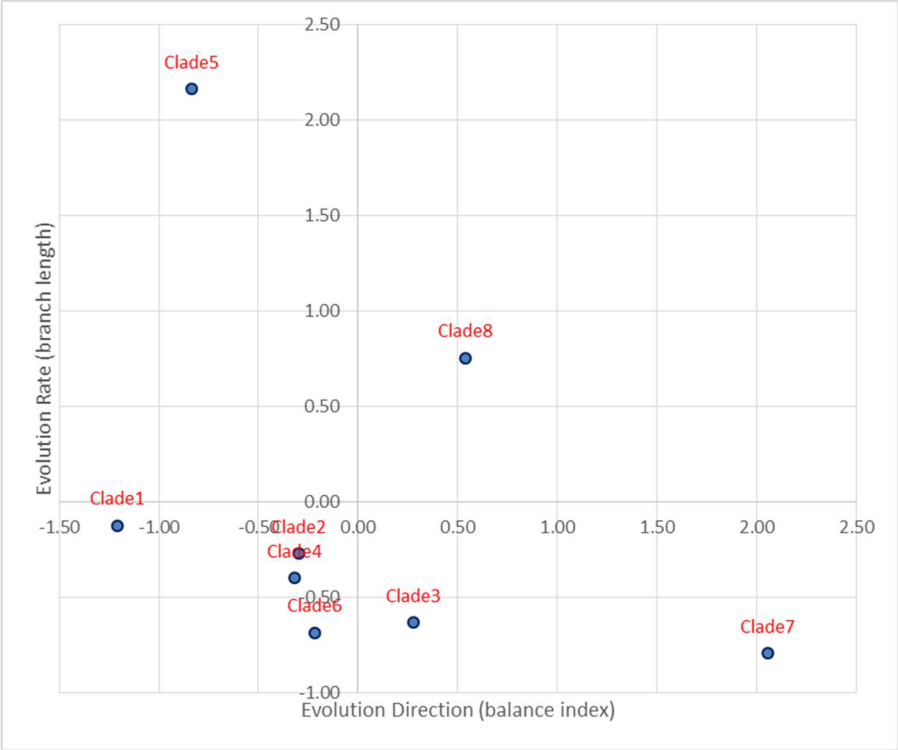

Figure 9. Plot of Different Clades on a Two-dimensional Space.

Table 13. Summary of Clades with Four Types of Generativity

	Uni-directional	Multi-directional
High Evolutionary rate	8	5
Low Evolutionary rate	3 7	1 2 4 6

The first dimension is what I refer to as evolutionary directionality, which indicates the direction of evolution of digital artifacts. Direction refers to whether they are evolving in a similar direction (i.e., uni-directional evolution) or diverse directions (i.e., multi-directional evolution). It has a similar idea to technological variety, as uni-directionality leads to less variety and multi-directionality leads to higher variety. A high balance index represents a more uni-directional evolution, while a low balance index represents a more multi-directional evolution. In my case, Clade 1 with its 1.6 balance

index represents a typical multi-directional evolutionary pattern, while Clade 7 with its 4.5 balance index represents a typical uni-directional evolutionary pattern.

Another dimension, which I call evolutionary rate, is measured by the average branch length. Evolutionary rate indicates on average how fast the digital artifacts within a group (i.e., clade) are evolving within a given time frame. For example, Clade 7 has an average branch length of 0.0009, which means on average each mashup only evolved 0.09% in terms of its defined genetic change. In contrast, Clade 5 has the highest evolutionary rate at 28%.

Table 14 summarizes the key differences for each type of generativity in my two-dimension framework. As I stated earlier, my thesis focuses on understanding the diversity in digital evolution. Without further support, I do not intend to argue which type of generativity is superior, particularly since companies may weigh the importance of evolutionary directionality and evolutionary rate differently according to their own considerations. My summary mainly serves as a reference to help API developers or platform owners find better strategies to promote their product based on their positioning objectives. For example, if Apple or WordPress want to increase the diversity of available apps or plugins, they should focus more on promoting competition among designers, such as by lowering entry cost. If the evolutionary rate is more important, encouraging diversified design is recommended, such as by offering challenges or awards and showcasing innovative and creative design. The Apple Design Award is a good example of this. This framework can also help web mashup developers to determine the best market to enter by analyzing existing mashups and environmental conditions and provide another assessment tool venture capital investors can use to evaluate a product's

chance of success. For example, a clone of a popular mashup may not necessarily be a bad choice when market competition is superseded by environmental forces such as end-users' preference. For example, many Clash of Clan clones are still very profitable despite the vast number of similar games.

Table 14. A Two-dimension Framework of Generativity

	Uni-Directional (Low Variety)	Multi-directional (High Variety)
High Evolutionary rate	Diversified design purpose in less competitive environment (example: when the Google Map API first became available, the competition was not so intense as a lot of different design ideas appeared that worked with Google Map)	Diversified design purpose and highly competitive environment (example: in today's mobile app market, the competition is so intense that only a few app designers can make profits; at the same time more and more novel apps are being created every day
Low Evolutionary rate	Similar design purpose and less competitive environment (example: most likely be seen in the casual market where mashups are designed based on individual designers' personal needs so that the mashups are quite similar to each other with minor tailored differences)	Similar design purpose and highly competitive environment (example: often seen in niche commercialized markets serving specific target customers)

Limitations

This paper is not without limitations. The base elements, defined as five categories of owner type, openness, service context, service function, license, are based on the logic composition of web APIs and mashups, whereas in biology, the defining characteristics of a protein are ultimately based on the physical composition of DNA.

DNA is actually the smallest and most basic structural unit of the gene, while my defined base units are not. This difference leaves room for further decomposition of my base elements. For example, I use service context and service function as base elements for defining the technical aspects of web APIs and mashups. However, this definition may be still too ambiguous regarding how it really works. This limitation is mainly due to the huge differences among web APIs and mashups, which use all kinds of programming language and structures. Future researchers could analyze digital artifacts at the source code level, which would decompose the elements at a finer level.

In biological applications that compare well-defined species, the results from the bootstrapping analysis are normally higher that what I report here, sometimes approaching close to 100%. While one might argue that relatively low numbers for the bootstrapping results indicate unreliable results, I believe that this shows the unique nature of the evolution of the digital technology. In biology, the major source of evolutionary change derives from mutation, where normally only small changes occur at a time. On the other hand, both mutation and recombination are major factors in digital evolution (Arthur 2009). Recombination provides radical changes at the gene level. When such radical change happens, the bootstrap value tends to be low. Therefore, my data may reveal the balance between mutation and recombination. As such, the higher the bootstrap value is (or the more significantly resolved the tree is), the more that mutation contributes to the variation. Conversely, the lower the bootstrap value is (or the less significantly resolved the tree is), the more recombination contributes to the variation.

In molecular biology, the phylogenetic analysis assumes that the change of each individual gene is random and independent from others' change. For example, the

mutation taking place in gene A should not be correlated with mutation taking place in other genes. However, in the case of web mashups, such an assumption may not still hold. One example is that the change of openness will be likely to trigger the change of licensing. Such violation of the biosocial assumption requires further detailed analysis of how much this may affect the result. The direct consequence of this type of violation is that the distance matrix used in the phylogenetic analysis should be developed from the actual observation of the probabilities of changes and co-changes. The distance matrix used in this study assumed the equal possibility of all changes. As I discussed earlier, though, this is not likely to be the real case. The Neighbor-Joining algorithm I used in this study is expected to generate close to correct result even with a questionable distance matrix or unknown evolution model (Mihaescu et al. 2009).

Conclusion

In this paper, I present a novel theoretical and empirical approach to study digital innovations based on modern evolutionary tools and theory. Digital innovations represent a new kind of innovation, often characterized by its rapid and frequent changes, due to its unique material properties such as programmability and data homogenization. Furthermore, since granular digital components can be easily combined to create new features, digital technology can create a far more dynamic and unbounded pattern of innovations. This poses a serious intellectual challenge to those who study innovations, as the existing innovation models do not adequately capture the highly dynamic and evolutionary nature of digital technology. I propose organizational genetics as a new theoretical and methodological approach to address this intellectual challenge. In particular, I propose a way to characterize the seemingly infinite varieties of digital

innovations I see, for example in web services, with a fixed set of genetic base elements. This allows us to understand underlying patterns across different types of digital innovations. In this study, I also demonstrate the efficacy of my organizational genetics approach through an exploratory study of the evolution of web APIs and mashups.

As web APIs become an increasingly important factor in the success of digital ecosystems, it is imperative to understand how changes in web services will affect the dynamics of innovations. By opening up the black box of web APIs and mashups and analyzing their evolution, I provide insight on this important domain. A multi-level analysis of mashups and web APIs that combines a sequence analysis with a subsequent phylogenetic examination is a promising first step to gain deeper insight into this important phenomenon that increasingly shapes our everyday experiences. This study is also a demonstration of how the organizational genetic approach can be applied in social science studies.

CHAPTER 3 ESSAY TWO - AN EVOLUTIONARY VIEW OF THE DEVELOPMENT OF OPEN SOURCE SOFTWARE PROJECT: THE CASE OF SOCIAL CODING IN GITHUB

Introduction

Open source software (OSS) has a long history that can be dated back to the early 1950s, when the source code of Arithmetic Language version 2 (A-2 system) was provided to users so they could suggest improvements. Starting in the late 1990s, the interest in OSS grew dramatically. Nowadays, OSS development has become an increasingly important and major approach to developing new software. There are more than 4.7 million OSS projects hosted on Github alone. Not only do individual developers make their own personal projects open sourced, but many companies also use OSS as their primary way to innovate. Facebook is considered the world's biggest open source company and even open sourced its core infrastructure technologies such as HipHop and Thrift.[6]

The success of OSS has drawn long-lasting academic attention as well. Lerner and Tirole (2002) used the word "startling" to express the surprise OSS made to economists. A large body of literature is devoted to understanding the motivation behind the volunteer work of OSS developers (von Krogh et al. 2012). Self-determination theory (SDT) has been widely used to explain individuals' motivation. Intrinsic and extrinsic motivations have proved to be two significant drivers for contributing to OSS (Hars and

[6] https://www.facebook.com/notes/facebook-engineering/folly-the-facebook-open-source-library/10150864656793920

Ou 2002, Lakhani and Wolf 2005). Another line of studies focuses on OSS development as a collective action using institution theories. These studies zoom out from the individual perspective and look at individual contributions in a social context. In recent years, with the rise of social media and the increasing popularity of distributed version control system in OSS, social coding makes the social context even more important in the development of OSS. Social media has greatly increased developers' awareness of the external environment, while distributed version control systems have greatly changed the traditional centralized governance during the development process.

However, little research has been done to understand the development process of OSS with these recent changes. Dabbish et al. (2012) highlighted the transparency brought by social coding and how has improved developers' inferences to their surroundings and further supported collaboration among developers. These prior studies mostly concentrate on the developer's action to understand OSS development rather than on the project itself. This can be largely attributed to the difficulty to track nuanced changes in each revision of OSS and make meaningful interpretations of these changes. However, it is critical to have a comprehensive understanding of the OSS development process in order to make better sense of the relationship between developer's contribution and OSS development.

To address this gap in the literature, I develop a computational approach to study the development of an OSS project through an evolutionary perspective. This approach is inspired by the evolutionary studies in biology to track the changes in DNA sequences. Specifically, I apply this approach to explore how OSS projects evolve. Based on my analysis, I propose a theoretical framework to classify different patterns of OSS

development based on their evolutionary patterns. By doing so, I seek to answer the following research question: How can we identify different evolutionary patterns of open source software by tracking down the changes made in each revision?

In the following sections, I first discuss prior research on OSS development and developer's contribution, followed by a discussion of the role of version control systems and social media in recent changes to OSS development. I then describe a case of an OSS projects and present my empirical approach. Based on my empirical study, I propose a two-dimension framework to classify the differences of generativity. Finally, I conclude with future directions and the overall implications of my study.

Literature Review

Open Source Software

OSS generally refers to software distributed with certain licenses, such as General Public License (GPL), that allow other users to review, modify and redistribute the source code of the software under specified conditions. After more than 20 years, OSS development has become an important means of innovation. Not only can individual users create and distribute personal software as OSS, but many large firms directly or indirectly adopt OSS as a way to enhance their competence. For example, Linux began as a personal project of then-student Linus Torvalds. Android was bought and open sourced by Google.

Scholars have been fascinated by the success of OSS. Many OSS programs have become the backbone of current digital life: Linux has been used by enormous companies such as Google, Twitter, and Amazon as their main servers. Android dominates the smart

phone market with over 82% market share. More than 50% of the world's websites are hosted with Apache servers. To understand the reason for the success of OSS, much attention has been put on the motivations of developers to work on OSS projects (von Krogh et al. 2012). The concepts of intrinsic and extrinsic motivation from self-determination theory (SDT) have been used to explain developers' voluntary participation and have been empirically tested. In a recent review, von Krogh et al. (2012) summarized four key factors covered in the literature: ideology (David and Shapiro 2008; Ghosh 2005; Lakhani and von Hippel 2003), altruism (Bitzer et al. 2007; Ghosh 2005; Hars and Ou 2002), kinship amity (David and Shapiro 2008; Hars and Ou 2002, Lakhani and Wolf 2005), and fun (Lakhani and von Hippel 2003, Lakhani and Wolf 2005). Career advancement and monetary incentive (Ghosh 2005; Lakhani and Wolf 2005) are considered the most important extrinsic motivations. Some scholars suggest that extrinsic motivation may be less influential than intrinsic motivation to encourage developers' participation (Har and Ou 2002).

von Krogh and his colleagues (2012) highlight the importance of social practice as a mutually constitutive factor with motivation to better explain developers' involvement in OSS projects. This type of research emphasizes the role of social influence in addition to individual motivation and argues that individual factors should be studied within their social context (von Hippel and von Krogh 2003). Some contextual factors that influence and mediate individual motivation include governance (Markus 2007; Shah 2006), community sponsorship (Shah 2006), the provision of rewards (Alexy and Leitner 2011; Roberts et al. 2006), and license restriction (Lerner and Tirole 2002).

Open Source Software Design

Another research stream, mostly from the computer science field, focuses on the software design itself from a more technical perspective. Studies in this stream address changes in OSS development, concentrating on the quality of OSS and code reuse in the OSS context. For example, rapid release is considered a favorable practice to improve OSS quality as it keeps developers interested and motivated (Aberdour 2007) as well as helps potential bugs be identified early (Raymond, 2000). Lee and Cole (2000) argue that code modularity can also help improve OSS quality because it allows individual developers to focus on single components at one time without interfering with other modules. This finding is supported by a similar study by Mockus and his colleagues (2002).

Related research also tries to identify various factors determining the success of software development. Some studies from a project management perspective use cost, time, and functionality as key criteria to assess success (Agarwal and Rathod 2005), while others from a user perspective rely on meta-level measurement such as user satisfaction, number of downloads (Crowston et al. 2003), number of users (Stewart and Ammeter 2002), and so on. However, these traditional assessments only work well for larger-scale OSS projects developed for more general use, not for smaller projects developed for niche use even though these smaller projects represent a larger portion of overall OSS projects. OSS projects normally do not have any well-defined milestones to achieve, so the conventional project management measurement cannot properly evaluate OSS projects, especially small personal projects. Similarly, many OSS projects are

initiated to meet the individual developer's own needs and thus may not draw massive attention or generate a high number of downloads. Therefore a new framework is needed to better understand OSS development by looking at the success of OSS from the perspective of process rather than outcome. Prior research on the development process either looks at developer activities or project changes like number of revisions and lines of code changes. Such measurements cannot capture the nuances among the revisions, and it is extremely difficult (if not impossible) to establish all the relationships in order to map each revision so as to gain the full picture of the evolutionary process.

Version Control in OSS

Version control software plays a key role in online collaboration in the OSS development process because it determines how distributed participants work together. When a group of developers work on the same software at different times and in different locations, version control software tracks the changes made by each developer and controls how each developer's change are applied to the original software version. It also enables developers to revert to earlier versions. In practice, there are two major types of version control systems: a centralized server-client system and a distributed version control system. The centralized version control system is more traditional and has been used for over 30 years. The most popular ones are the Concurrent Versions System (CVS) and the Apache Subversion (SVN), which have been used in many larger projects such as Apache, Python and Ruby. SourceForge, one of the biggest online repository hosting platforms, also uses SVN for projects it hosts. For projects using a centralized version control system, there is only one central repository where every developer needs

to “check-out” and “check-in” in order to make edits to the central repository. Therefore, there is normally an administrator to manage the collaborations among these developers to make sure no versions conflict.

The distributed version control system is newer. It can be dated back to the early 1990s and is especially used in Linux development. However, it did not gain major popularity until the twenty-first century. As the fast-growing Internet brought together developers from all over the world and the open source community grew, the distributed version control system started to surpass the centralized system. Unlike the centralized system, it does not require a centralized repository to keep the current software version. Instead, it allows individual developers to have their own copy of the working repository by copying from others and to work remotely and even offline. With such distributed characteristics, developers have much more freedom to implement their own design goals without interfering with other people. In addition, it seems to improve the quality of OSS. In a case study on the Linux project, Moon and Sproull (2000) found that distributed development allows developers to work on the same issue simultaneously in a competitive manner that increased the probability of providing better quality solutions. Git, originally launched in 2005 for the Linux project, is the most popular version control software used in OSS development. It was reported[7] that more than 42% of developers use Git as their primary version control system.

[7] Eclipse Foundation report is based on a survey of its developer community. https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/

Social Coding and OSS Development

Open source projects represent a new form of organizing for knowledge creation through a global digital infrastructure (Faraj et al. 2011). Open source communities have emerged as an informal mechanism to facilitate knowledge exchange and rapid diffusion of innovations across conventional organizational boundaries (Chesbrough, Vanhaverbeke, and West, 2006). In such forms of organizing, individuals who do not necessarily belong to the same formal organization can temporarily work together through digital technology. Among many benefits of such forms of organizing, one that has not been explored systematically is the fast evolution of digital artifacts. Digital artifacts (i.e., software codes) developed in open source communities do not remain as they were originally created. Many open source codes become the basis of new projects, while others are adopted to solve unique local problems. These artifacts, thus, continuously evolve as they are being diffused throughout the community. The rising popularity of distributed version control systems such as Git in open source project development makes such diffusion even more generative. For example, Gousios and his colleagues (2014) found faster turnaround, increased engagement and decreased response time in OSS development on Github. Jiang et al. (2013) also found that the contributors of OSS projects on Github were widely distributed in the developer communities and that projects can reach developers far away from the original project creator.

Social coding combines the traditional coding activity in OSS communities with social media, which allows developers to track and follow the activities of other developers or projects (Dabbish et al. 2012). It is a recent movement in the open source

community that is becoming popular thanks to the rise of the Internet and distributed version control systems. Tom Preston-Werner, founder of Github.com, said that social coding essentially "takes the ideas of a social network and add on top of that code hosting"[8] For example, Github.com, the most widely used project development hosting platform using Git, offers functions that allow users to follow other users, track public projects, and discuss them with others in the developer community. This unique combination of distributed development and social networking has changed the way open source projects are developed. The distributed nature of the Git system enhances the independence of the developers, who no longer need permission from the administrators of the project in order to make any modification, making it extremely easy for them to make a separate branch and experiment with new ideas. These developers can be socially connected to other developers and monitor any public activities done to the project. Such transparency increases the awareness of developers.

To address the lack of understanding of OSS development in the context of the emerging social coding phenomenon, I propose an evolutionary view to focus on the process of OSS development. In this study, I am particularly interested in how to identify the evolutionary patterns and classify OSS programs based on their evolutionary patterns.

[8] http://www.sitepoint.com/podcast-107-social-coding-with-githubs-tom-preston-werner/

A Computational Approach to Analyze the Evolutionary Patterns of OSS Projects

Evolution of Open Source Software

There are two basic modes of evolution from a molecular biology standpoint: recombination and mutation (Spears 2000). Recombination mainly concerns exogenous changes in which genes from different sources may be mixed to create a new combination of genes to reproduce the next generation, like in the case of allogamy. This process is considered one of the most important sources for genetic variation. By contrast, mutation mainly concerns endogenous changes when certain genes are randomly alternated, such as through deletion, duplication, or even substitution. As Arthur notes (2009), similar phenomena can be observed in technology evolution including OSS project developments. Recombination often takes place when source codes from other projects are reused and integrated to initiate a new project. After the project is launched, mutation plays an even bigger role as developers keep modifying the source code with incremental changes.

Fleming and Sorenson (2004) suggests that innovation can be conceptualized as a combinatorial search process in which science can be used to highlight useful combinations and eliminate wrong paths. In the context of OSS, Lee and Cole (2003) claim that OSS project development can be seen as an evolutionary process driven by criticism and error correction. Here I also argue that changes in source code need to be seen as the main engine of OSS project development. Especially in social coding, individual developers may take different revision directions that greatly increase the search space. Thus, even though sometimes the changes in source code are eventually

revoked, they still provide valuable feedback to help the project move to better solutions. In modern biology, variation caused by genetic change is the foundation of evolution in which selection then determines survival direction. In OSS, particularly in the case of social coding, variations of projects through changes in source code created by developer communities become the basis of advancing the project. The adoption of these codes by users can be likened to selection. In this study, we focus on the causes and patterns of variations in OSS projects.

Organizational Phylogeny

To understand how an OSS project evolves as its source code is modified by different developers over time, I employ a biology-inspired analytic approach to study the evolution of source code. We call it organizational phylogeny. This approach uses common evolutionary genetic techniques, particularly sequencing and alignment based on phylogenetic analysis, to better understand my data. Existing research does not discuss how OSS codes evolve over time as they are reused or how software codes developed from one project can contribute to other projects. Prior studies on OSS development have not been able to explore how the source codes evolve and produce different derivative innovations as a way of assessing the success of an OSS project. Phylogenetic analysis is a comparative method widely used in biology to extract information about the evolutionary relationships of organisms and to trace the changes along the evolutionary process. We have chosen this approach because of the similarity between raw source code and DNA sequences, which are basically a string of sequential elements. We consider each letter, number and symbol written in source code equivalent to nucleotides

in DNA sequence. While a DNA sequence consists of four nucleotides (represented by C, G, T and A), a source code consists of a limited number of ASCII characters that are allowed in software source codes. In this study, we treat project source code as a genetic sequence of the project, the characters, numbers and signs in the source code represents the genetic building blocks, or the software DNA. The code from different versions or branches can then be aligned and analyzed in a comparative manner as is done in phylogenetics. Phylogenetics is a subject in biology that specialized in studying the evolutionary history of species. In modern biology, especially molecular biology, it primarily relies on computational approaches to analyze the DNA sequences in order to infer the relationships among different species. It compares comparing the (dis)similarities of DNA sequences from different species and generate the phylogenetic trees to represent the evolutionary relationships of the species. The phylogenetic tree then can be verified by comparing it with known fossil records. Similarly, my approach is verified with known revision history to make sure the ancestral relationships are indeed correctly captured through this method.

The first step in a phylogenetic analysis is to align each source code to generate an alignment score as well as an alignment matrix. The purpose of the alignment is to standardize the source code across different projects for easy comparison. Alignments are performed using software called ClustalTXY, a derivative of Clustal, which is an alignment software program commonly used among genetic researchers but adapted here for social science studies. This software performs pairwise alignment of sequences and constructs a pairwise similarity matrix. The result of the alignment analysis provides a foundation to analyze the differences and relationships between sequences. Figure 10

shows the alignment results for a short segment of code. It shows a small portion of this segment is different: only the first five elements (From PPORT to -BIND) are different between top 9 sequences and the bottom 7 sequences.

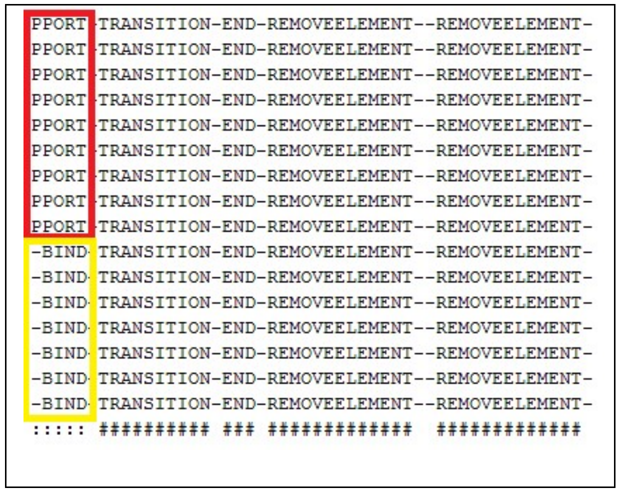

Figure 10. Example of Alignment Results in One Section of Code.

After generating the alignment results, ClustalTXY also provides phylogenetic analysis results in Newick format, which contains information regarding branch length and relationship. We then import the results into visualization software (we used Treeview, another popular program) to generate a phylogenetic tree. Phylogenetic trees allow us to infer the relationships of each sequence relative each other, where each sequence represents a taxon (e.g., a species in biology) or a version code (this project). A typical phylogenetic tree (shown in Figure 8) has two main elements, node and branch. A node (such as n2 and n3) refers to taxa (biology) or a version of source code (this project). The root (n1) is special and refers to the oldest extant node among all taxa in the tree. A rooted phylogenetic tree means a root is identified and all other taxa can be positioned based on the root. In biology, a root represents the common ancestor of all taxa in the tree and is normally identified based on the fossil record. In this study, the root

represents the earliest source code that is identified through my data. The branch length depicts the evolutionary distance between two taxa or differences between two versions of source code. A distance scale provides a reference to understand the magnitude of the branch length. For example, a branch with length of 0.004 in the example tree means a 0.4% difference in the sequence. Therefore the difference between n1 and n3 in Figure 11 is roughly 0.8% (the total length of branch 1 and branch 2). Together, the overall structure of the tree, or the topology, offers a visual representation of all the taxa or versions.

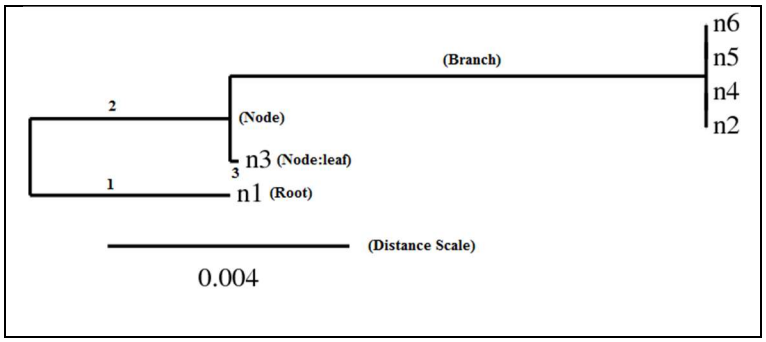

Figure 11. Notations Used in Phylogenetic Tree.

In the example in Figure 8, n2 to n6 are five different versions of files released sequentially, all originating from n1. This example phylogenetic tree tells us that version n3 is most similar to n1, although it is released after version n2. Versions n2, n4, n5, and n6 are almost the same, suggesting that version n4 made reverse changes to n3 and reverted to n2 and subsequent versions (n5 and n6) retained n4's changes. The difference between n1 and n3 is 0.8%, suggesting that 0.8% of the source code has been changed in n3 when compared with n1. We can tell the revision history of the example project from this tree: after the initial release of version n1, revision n2 made some big changes, then in revision n3, some changes in n2 were reverted, making n3 more similar to n1 than n2.

Afterwards, revision n4 went back to n2's changes and only very minor changes were made in the n5 and n6 revisions.

Measuring Evolutionary Directionality and Rate of Change

After the phylogenetic tree is built, we can then infer the evolution details from the tree. An important property of tree shape is balance. Figure 12 below shows two extreme examples of trees with different balance. In general, tree balance measures how subgroups from the tree root point are distributed. For example, tree A in Figure 12 gives an example of a perfectly balanced tree in which every speciation point, including the root, has the same number of branches at left and at right. Tree B gives an example of a perfectly imbalanced tree in which every speciation point, including the root, has only one taxon at left and all others at right. Tree balance is often used by evolutionary biologists to infer macro-evolutionary phenomena regarding speciation. In the context of software development, Linux can be seen as a more balanced example than Microsoft Windows. Linux is a family of operating systems with different versions designed for multiple purposes, such as desktop use, tablet use, embedded system, and server, with many different distributions such as Ubuntu or Red Hat, all of which are evolved from the same ancestor.

In my study, I follow the same approach to analyze the "speciation" of OSS projects so that I am able to understand how and why different versions emerge. Therefore, I use tree balance as a measurement of evolutionary directionality. A more balanced tree suggests that the OSS project has been evolving in multiple directions, while a less balanced tree suggests the project has been evolving in a similar direction.

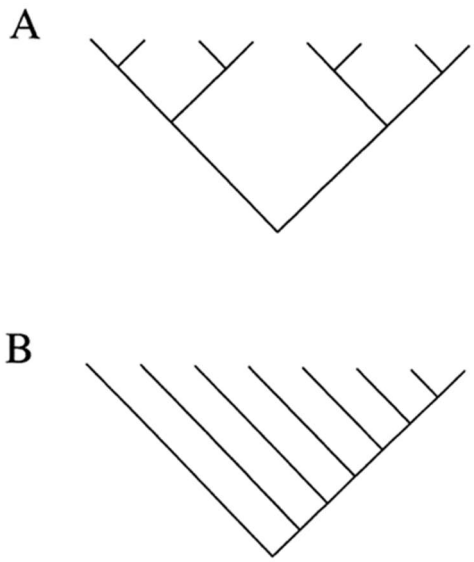

Figure 12. Example of Tree Balance.

Case of Open Source Software Development

Open Source Projects on Github

I collected my data from Github, one of the largest online hosting repositories for programming code. Git is a widely used distributed revision control and source code management system that allows users and developers to track every single modification of their software source code. As such, it provides a perfect research context for understanding how innovations diffuse over time. In addition to tracking the project's own source codes, Github also allows developers to easily "fork" other developers' projects. Forking is an action in Git which basically refers to the way one developer can clone or save one existing project owned by other developer into his/her own repository so that he/she can use and modify it as a separate project.

jQuery Plugins

There are many different projects on GitHub, but we chose to focus on jQuery plugin projects. JQuery plugin is a light plugin created to extend JQuery's functionality using JavaScript. Given the differences (syntax, structure, and etc.) among the kinds of programming languages used in GitHub, it is extremely difficult to compare projects written in different languages. Focusing on projects written in the same language allows us to make comparisons while controlling other language-associated confounding factors. The focus on jQuery plugins allows us to only focus on JavaScript language only. We also choose JavaScript because it is the most popular programming language used on Github, representing more than 27% of projects hosted there (Bissyande et al. 2013). Furthermore, the code structure of jQuery allows us to calculate the measurements of the evolution of digital artifacts and compare them among different projects. In prior studies (Ray et al. 2014) on the effect of programming languages on software quality, process factors such as project size and commit numbers overwhelmingly dominate the effect of programming language type. Therefore, it is reasonable to believe the result from jQuery projects will still hold for other types of projects.

Currently, there are 21,315 jQuery plugin repositories on Github. We collect data from projects that match the following criteria. First, the project has to be created after February 12, 2012. Since Github only published data starting from this date, doing so allows us to track any changes to a project so that I am able to create a meaningful evolutionary pattern for the project. Second, I only include projects that have at least 30 forks in order to have enough nodes and edges to create a network. There are three

projects in our data set that have abnormally large developer networks because a single developer forked a huge amount (more than 13,000) of projects; these are excluded from our final analysis as outliers. Table 15 summarizes my data set of the 146 projects matching the criteria and included in this study.

Table 15. Summary of Project and Networks

	Project Attributes		Developer Attributes	
	Number of Revisions	Age of Projects (in months)	Average Number of Forked Projects	Average Number of Followed Developers
Max.	933	38.7	1073	880
Min.	5	14.3	1	1
Mean	88.9	25.8	23.60	4.58
Median	44	25.1	12	2
SD	137.67	6.8	51.90	24.37
$N = 146$				

I first demonstrate organizational phylogeny with two examples. jQuery-Box-Slider is a project created and primarily maintained by developer P-M-P. It is a popular project designed help website developers to easily create slide animation. It has gained more than 215 forks since it was launched in May 2012. I collected all 100 different versions of the source code and present their phylogenetic tree in Figure 13. jQuery-Fast-Click is a project created and primarily maintained by developer dave1010 to provide a simple workaround fix for mobile browser delay. It has more than 41 forks and 117 stars since it was launched in March 2011. Figure 14 shows jQuery-Fast-Click's phylogenetic tree with all 27 versions of its source codes.

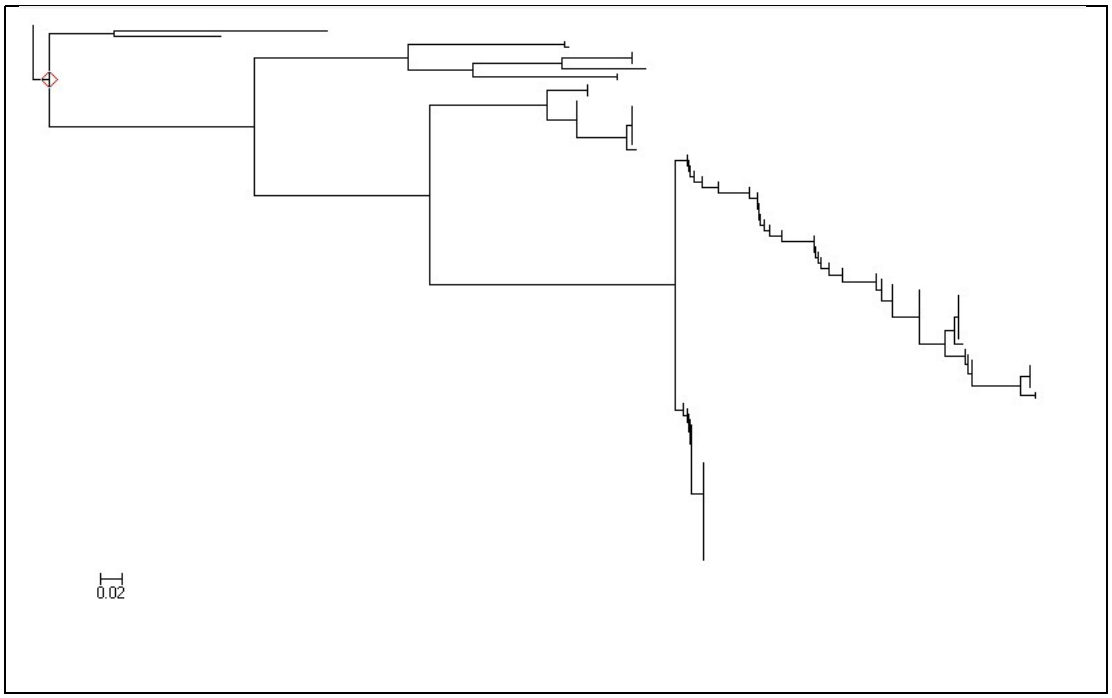

Figure 13. Phylogeny of Different Versions of "jQuery-Box-Slider" Sequence Code.

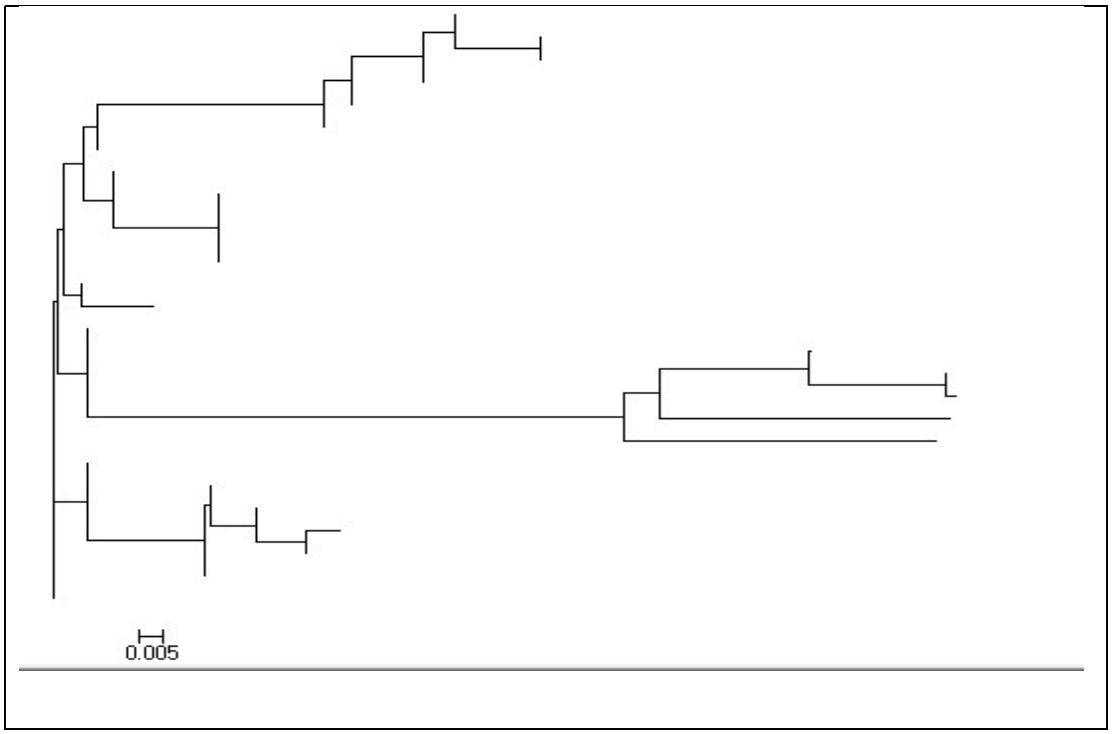

Figure 14. Phylogeny of Different Versions of "jQuery-Fast-Click" Sequence Code.

To quantify the directionality of evolution, I adopt a mathematical instrument called Colless Index (Colless 1980), a widely used measurement of tree balance in phylogeny. This measurement is done by inspecting each internal node and comparing the number of tips on each side. In my study, I refer to this as the directionality index. The higher the directionality index, the more similar the versions of the source code are and vice versa. I also calculate the branch length within each clade. Calculation was done using APTreeshap in R (Bortolussi, Durand, Blum, and Francois, 2005). In the following paragraphs, I illustrate how I infer evolutionary rate and direction based on this phylogenetic approach in detail, using data from the jQuery-Box-Slider and jQuery-Fast-Click projects.

Directionality of evolution can be measured by the directionality index. For jQuery-Box-Slider, the directionality index of 31.16 suggests a highly unbalanced tree shape. This means that most of the later source codes are built on a few earlier versions. In Figure 15, I show the dendrogram of the tree to display the hierarchical structure. By ignoring the branch length, a dendrogram lays out every version of the source code and reveals which versions overlap due to minimal difference in the phylogenetic tree, emphasizing the relationships among different versions of source code. It is clear to see there are two major groups of source codes. Within each group, a newer version of the source code has slight changes (hence “overlapped” in the phylogenetic tree) over previous versions along the same direction. To compare, I show a dendrogram of jQuery-Fast-Click, which has a balanced index of 0.85, in Figure 16. In this tree, there are more diversified groups of source codes. Unlike jQuery-Box-Slider, it has more variations from the different early versions.

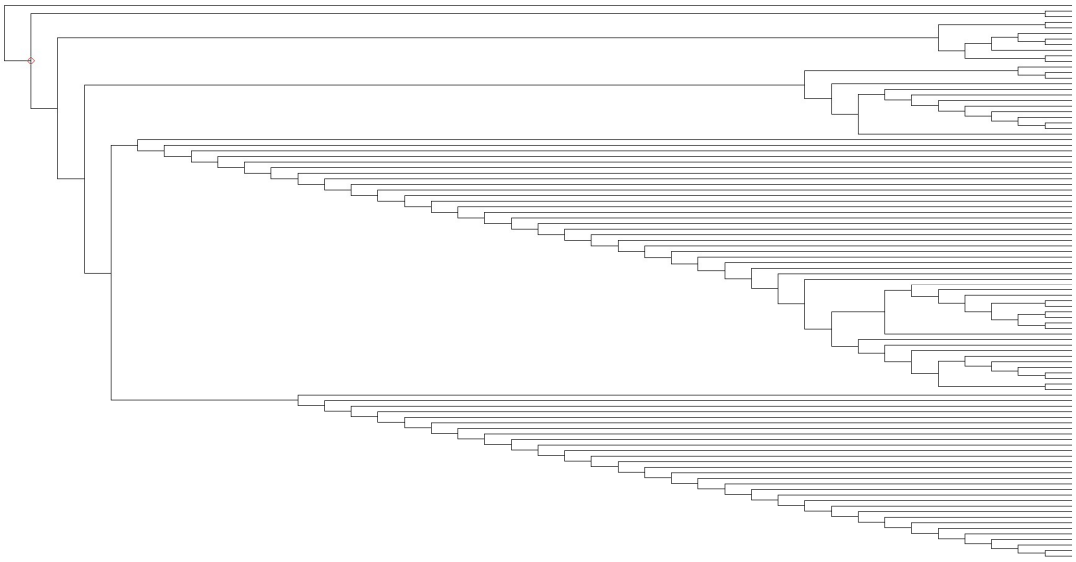

Figure 15. Dendrogram of Different Versions of "jQuery-Box-Slider" Sequence Code.

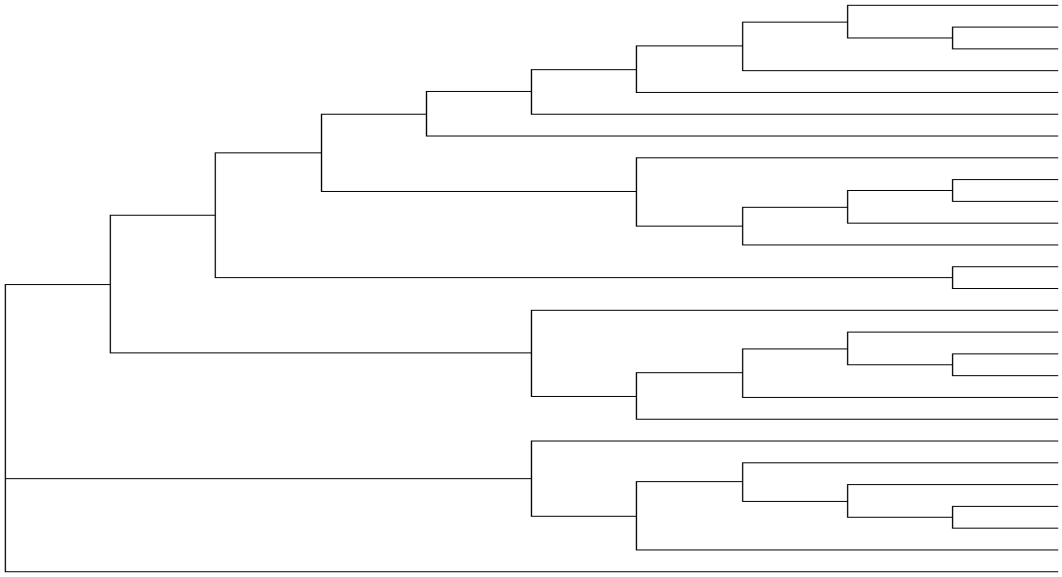

Figure 16. Dendrogram of Different Versions of "jQuery-Fast-Click" Sequence Code.

I calculate the rate of evolution by dividing the total branch length by the age of project. The branch length represents the differences among sequences. In this case, it represents how many changes have been made. I consider changes in source code as a

physical representation of development in ideas. By adding all the branch lengths together, I get the sum of lengths for the tree. In the case of jQuery-box-slider, the total branch length is 1.95, divided by 24 (months), for an average branch length per month (i.e., rate of change) of 0.08. That is, on average during the 24 months after this project was launched, the source code changed at roughly 8% per month. To contrast, jQuery-fast-click changed at roughly 1.1% per month over 38 months.

I believe this analytic approach is an appropriate way to answer my research questions. First, in my study, I will perform this analysis at the scale of hundreds of branches across multiple months or even years. In doing so, I can create a tree based their similarity and differences with time information. The overall structure of the tree, or the topology, offers a visual representation of all the taxa or versions. Different structures represent different evolutionary patterns (Kirkpatrick and Slatkin 1993) of the direction and speed of evolution. Secondly, I can not only create a qualitative tree but also calculate quantitatively in terms of percentage change and percentage change over certain time to assess the proposed two dimensions of evolution. The quantitative measurement then provides a foundation to perform further analysis of how OSS project development may be affected by other factors.

I conduct this analysis for each project in my dataset. Table 16 summarizes these two evolution dimensions. As the balance index summary shows the mean of 6.53 and median of 3.65, it suggests that projects in my data set in general are evolving in a similar direction, which is not surprising in the context of open source project development.

Table 16. Summary of Two Dimensions of Evolutionary Pattern

	Evolutionary directionality	Evolutionary rate
Max.	31.16	10.75%
Min.	0	0
Mean	6.53	2.99%
Median	3.65	1.72%
Std. Dev.	7.52	2.93%
$N = 146$		

Clustering Analysis

Based on my two-dimension framework of classifying evolutionary pattern, I then performed a K-mean clustering analysis with 4 as the designated number of clusters. The specification of K in K-mean as 4 was primarily driven by the theoretical framework of 4 potential different evolutionary patterns. Table 17 shows the properties of each cluster. The clustering result supports my theoretical 2 by 2 framework. If clusters 1 and 2 are low in balance index and 3 and 4 are high in balance index, then clusters 1 and 3 are high in evolutionary rate and 2 and 4 are low in evolutionary rate. The differentiation was statistically significant and supported by ANOVA test (Tables 18 and 19).

Table 17. Summary of Four Clusters of Evolutionary Patterns

Cluster	Number of Projects	Balance Index	Evolutionary rate
1	38	3.23	4.76%
2	68	2.86	0.84%
3	19	17.65	9.10%
4	21	17.16	1.68%

Table 18. ANOVA Analysis of Balance Index by Clusters

	Mean	Lower 95%	Higher 95%
Cluster 1 and 3	3.00	1.62	4.388
Cluster 2 and 4	17.38	14.96	19.81
p-value < 0.0001			

Table 19. ANOVA Analysis of Evolutionary Rate by Clusters

	Mean	Lower 95%	Higher 95%
Cluster 1 and 3	0.06	0.0529	0.0670
Cluster 2 and 4	0.01	0.0045	0.0159
p-value < 0.001			

Discussion

Based on my analysis, I am able to identify evolutionary direction and evolutionary rate as two important dimensions to examine the evolutionary process of OSS development. Below I will discuss how different evolutionary patterns may emerge as the result of environment pressures.

Competition and OSS Evolutionary direction

Phylogeny has helped biologists to infer the evolutionary process based on the genetic similarity between different organisms. In biology, ecologists are particularly interested in understanding how speciation took place. I am interested in a similar process, that is, how OSS branching takes place. External factors such as environmental forces are widely considered by ecologists to be the main force that directs evolution. Scholars (Davies et al. 2012; Webb et al. 2002) have proposed environmental constraints and competition as two primary forces, and tree shape can be altered by the changes of these two forces. The domination of the two forces can be easily detected by the tree balance index: the tree balance/imbalance is greatly affected by the balance between competition and environment. When the environment is stable and competition becomes the primary evolutionary force, the tree will become more balanced because competition would restrict the coexistence of species with similar traits (Davies et al. 2012; Grant

1972). As species with similar traits exist in the same environment, they would inevitably compete for similar resources; as a result, reducing competition may lead to different trait paths that avoid competition, generating diversified evolutionary patterns. Therefore, it is more likely to have multiple branches at each splitting node, resulting in a balanced phylogenetic tree. In contrast, when the environment is highly volatile and the fit to the environment becomes more critical for survival, species would be expected to share more similar traits (Webb 2000; Webb et al. 2002), which means the evolutionary pattern would be more uni-directional. Environmental volatility essentially prohibits other divergent evolution paths that do not fit, leaving surviving species more alike.

I theorize that OSS will show highly divergent patterns of evolutionary directions when facing volatile and uncertain market and technological conditions and more convergent patterns when facing a stable environment with stiff competition among similar products in the market. This is consistent with Baldwin and Clark(2000), who note that when the environment is stable, industries become more vertically integrated with few variations across competitions, while in a volatile environment, firms often pursue a "mix and match" strategy to satisfy changing market demands, which leads to a fragmented industry structure. Therefore, I propose:

PROPOSITION 1: When market competition is low, digital artifacts will evolve in the same direction; when market competition is high, digital artifacts will evolve in divergent directions.

In the specific context of open source projects, I consider cross-project competition as one aspect of market competition. Cross-project competition concerns the amount of similar projects. For example, there are quite a few jQuery plugin projects

related to the swipe function but only a couple of projects for snapshot. For one individual project, the existence of a large number of similar projects creates a tough environment which can force it to find a niche area and evolve in an exploitative way in order to survive. Within the project, the relationships among developers (i.e., cross-developer competition) also influence the evolution of project. When the developers communicate well with each other, they tend to achieve a consensus. Lacking communication, they are more likely to try out different modifications. This is especially true in the decentralized OSS development on Github.

To support my proposition, I did a test using data from Github. To measure the market competition, I collected the total number of similar jQuery projects for each project in my dataset by using the functionality keywords to search repositories on Github. Table 20 summarizes the statistics of the market competition measurements.

Table 20. Summary of Market Competition Intensity for Each Project

	Number of Similar Projects
Max.	350
Min.	1
Mean	58
Median	14
Std. Dev.	93

ANOVA test shows that there is significant difference between balanced projects (projects in cluster 1 and cluster 2, which tend to evolve in divergent directions) and imbalanced projects (projects in cluster 3 and cluster 4, which evolve in similar directions). As shown in Table 21, on average, imbalanced projects have more than two times the competition (104.4 similar projects) of balanced projects (38.5 similar projects).

Though it is not sophisticated, my result provides potential evidence of the effect of market competition on the evolutionary direction of digital artifacts.

Table 21. ANOVA Result of Market Competition by Evolutionary Direction

	Evolutionary direction Mean	Market Competition Mean
Cluster 1 and 2	3.00	38.5
Cluster 3 and 4	17.38	104.4

Competition and OSS Evolutionary rate

The second aspect of the evolutionary dynamics of OSS projects I studied is the evolutionary rate. Evolution speed is a major topic in the fields of biology and management. Various factors have been proposed to explain the evolutionary rate. Environment has been considered a particular important factor in both the natural and social sciences. For example, using computer simulation, Kashtan and his colleagues (2007) found that varying environment may significantly speed the process of evolution in nature. Similarly, Worden (Worden 1995) found that the speed of evolution is constrained by the pressure of natural selection from the environment. Prior research on innovation and new product development has similar findings in that it is generally considered that greater competition can drive faster innovation speed (Emmanuelides 1991; Kessler et al. 1996; Pearce 2002; Porter 1990). This is not surprising as faster speed can help companies to establish a first-mover advantage, setup industry standard and lead the competition. Therefore, I propose:

PROPOSITION 2: An environment with intense competition will lead to a faster rate of evolution.

I did a similar test for Proposition 2 as for Proposition 1. As shown in Table 22, the significant ANOVA test result indicates that the faster evolving projects (cluster 1 and cluster 4) are facing almost five times more competitors than slower evolving projects (cluster 2 and cluster 3)

Table 22. ANOVA Result of Market Competition by Evolutionary Rate

	Evolutionary rate Mean	Market Competition Mean
Cluster 2 and 3	1% per month	22.43
Cluster 1 and 4	6% per month	107.00

Four Types of Open Source Software Development Patterns

Evolution as a metaphor has been used by many previous researchers to study different types of change over time. "Unprompted change" is the key in the notion of generativity defined by Zittrain (2006). From the example phylogenetic tree in previous section, I can see two aspects of change in evolution: one aspect concerning the degree of changes, which is represented in branches (the longer the branch, the bigger the change), and the other concerning the variety of change, which is represented in nodes. Therefore, in this study, I unfold the notion of evolution and understand it as a two-dimension concept. Together these two dimensions can give a comprehensive view of the evolutionary patterns in terms of where and how the evolution is towards to. Similarly, I am also interested in two related factors in the evolution of digital artifacts such as OSS project development. The rate of change in OSS concerns how fast the source code is being modified. Faster changes can sometimes mean a more agile style of development to adjust to external requirements with a potential higher chance of issues, while slower changes may suggest a more conservative development style with stability as the primary

concern. Directionality in OSS refers to how a new version of a project is developed regarding variety. Is it following the same direction as previous versions, which means a more exploitive development with low diversity, or in an exploratory manner that may create a more diversified project family? These concepts form four potential evolutionary patterns as illustrated in Figure 17.

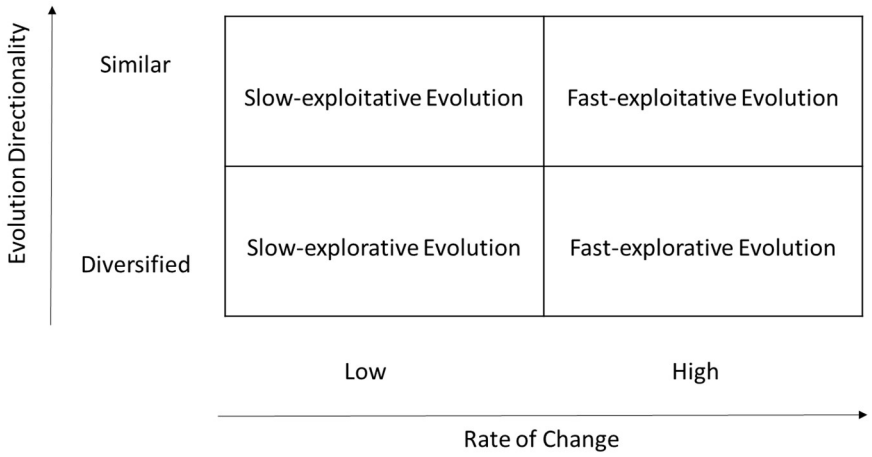

Figure 17. Two-dimension Framework of Evolutionary Pattern.

The primary purpose of the study was to explore different types of generative digital artifacts. Generativity is uniquely marked by "unprompted change" (Zittrain 2006), therefore I argue that evolutionary patterns can be used as a meaningful way to classify different generative digital artifacts. From my analysis, I propose directionality and rate of evolution as two primary dimensions that characterize the evolutionary patterns of generative digital artifacts. Thus, I tried to identify four major different types of generativity with different evolutionary patterns.

The first dimension is what I refer to as evolutionary directionality, which indicates the direction of the evolution of digital artifacts, that is, whether they are

evolving in a similar direction ("uni-directional evolution") or in diverse directions ("multi-directional evolution"). Uni-directionality leads to less variety and multi-directionality leads to more variety. I argue that environment is a key factor that can drive evolutionary direction. In addition to the market competition I proposed earlier, there are also other potential environmental aspects. For example, market need can potentially influence evolutionary direction. When the market need is simple, the digital artifacts are more likely to be similar in order to respond the simple need. In contrast, when the market need is diverse, the digital artifacts are also more likely to evolve differently to meet various needs.

The second dimension, which I call evolutionary rate, is measured by average branch length. Evolutionary rate indicates on average how fast the digital artifacts are evolving within a given time frame. Environment also plays an important role in determining evolutionary rate. Earlier I demonstrated that competition intensity can speed up the evolutionary process. Other factors, such as technological complexity, can easily affect the evolutionary rate.

To summarize, my framework suggests that there are four different types of generative digital artifacts depending on the way they evolve. Table 23 summarizes key differences among these four. Without further support, I do not intend to argue which type of generativity is most superior, particularly since companies unconsciously weigh the importance of evolutionary directionality and evolutionary rate differently according to their own considerations. My summary mainly serves as a reference to help designers of digital artifact to find better strategies to promote their product based on their needs.

Table 23. A Two-Dimension Evolutionary Patterns for OSS

	Uni-Directional (Low Variety)	Multi-directional (High Variety)
High Evolutionary Rate	Medium generative digital artifacts that are rapidly evolving along the same path (example: jQuery-Countdown has 196 competitors providing similar functionalities). They are more likely to be seen in highly competitive markets.	Highly generative digital artifacts that are rapidly evolving towards different directions (example: jQuery-Truncate with 73 competitors). They are more likely to be seen in less competitive markets.
Low Evolutionary Rate	Low generative digital artifacts that are slowly evolving along the same path. (example: jQuery-i18n with 49 competitors). They are more likely to be seen in less competitive markets.	Medium generative digital artifacts that are highly evolving towards different directions (example: jQuery-etherpad-lite has no competitors). They are more likely to be seen in very low competitive markets.

Limitations and Future Research

Of course, this work is not without its limitations. First, I primarily focus on jQuery plugin projects in my study. Their relative simplicity makes my analysis feasible within my computational capability, but complexity can play a role in determining the evolutionary pattern. However, the two-dimension framework of generativity will still hold even in more complex projects. Future research with more sophisticated analytic methods can include more complex projects using different programming languages and test how complexity can actually influence these two dimensions.

Second, my research analyzed the evolution of digital artifacts through the changes of each released versions of individual project. Therefore, the primary change taking place is mutation, or incremental changes over time. With digital technology, however, both mutation and recombination are major factors in digital evolution (Arthur

2009; Brynjolfsson et al. 2014). Recombination can precipitate large innovative change. In OSS development, recombination is a popular way to initiate new projects and introduce new functions. Future researchers can conduct cross-project analysis to examine how different projects may be interconnected and co-evolve over time.

Third, as in the first essay, the basic assumption of this phylogenetic analysis on the independence of mutation is violated to some extent. The revision of source code is normally performed at the level of words or even the level of phrases, instead of the level of characters. However, since I use the raw source code as the equivalent of DNA sequence, the actual level of analysis is the character in this study. Therefore, some changes will be correlated. This difference leads to two issues: first, the changes detected by the software will be overestimated compared to the actual change. For example, changing from "private" to "public" is treated as a 6-character change and a 1-word change, and change from "if" to "for" is treated as 3-character change and a 1-word change. The degree of change will be different when the analysis is done at the character level but the same when the analysis is done at the word level. But such difference is universal and not biased to any certain type of revision, therefore my result should still be close to the actual result. This issue also brings opportunities for future researchers to develop better tools to handle analysis at higher levels. The second issue is that the distance matrix should be based on the observed data rather than equal probability. Though the Neighbor-Joining approach is good at dealing with this kind of violation, a more accurate result can be achieved by calibrating the analysis with actual observed data. Future researchers can focus on the creation of such a distance matrix by measuring the change frequencies in software development. It took biologists several decades of

hard work to build reliable evolution models, but this will be an exceptionally rewarding work that create the foundation for all kinds of further researches.

Forth, due to the scope of the paper, I cannot fully explore potential factors affecting the evolutionary process. With limited data, I conducted simple tests to provide some evidence for my proposition to explain the effect of market competition. A more sophisticated and rigorous analysis controlling for other possible factors is needed. There are also many other factors, such as designer's characteristics, technological complexity, market needs, and etc.

Conclusion

Digital innovations represent a new kind of innovation, often characterized by its rapid and frequent changes, due to its unique material properties such as programmability and data homogenization. Furthermore, since granular digital components can be easily combined to create new features, digital technology can create a far more generative and unbounded pattern of innovation. This poses an intellectual challenge to those who study innovation, as existing models do not adequately capture the highly dynamic and evolving nature of digital technology. The analytic approach in this essay provides new theoretical and methodological approaches to address this intellectual challenge and allows us to understand underlying patterns across different types of digital innovations. I demonstrate the efficacy of my approach through an exploratory study of the evolution of OSS projects on Github. To the best of my knowledge, this study is the first to quantitatively measure and identify evolutionary patterns and create various future research opportunities.

My study has important implications for managers, who can benefit from knowing what characteristics are more likely to make a project generative. Policy makers and governing bodies of open source projects can use the findings of this study to formulate best practices such as code reuse to improve the generativity of a platform. The findings also provide suggestions to better regulate OSS development. Many people only focus on the bright side of openness and ignore the dark side. Tor is a classic example of how OSS can be wrongly used by bad people. By evaluating the current evolutionary pattern of OSS projects, my framework can help to predict the possible future of OSS and thus support identification of potential bad software. For example, a project with very slow evolutionary rate and safe uni-directionality, such as weight loss monitoring, can be viewed as a low-threat OSS, in comparison to a high risk project with very fast evolutionary rate and unsafe multi-directionality, such as network traffic monitoring.

Researchers interested in OSS development, particularly in code reuse, have been using computational techniques such as file comparison tools to analyze how certain code may transfer between projects. Many of these studies are conducted by computer scientists, whose attention has been put on how to identify and improve the code block reused most. Few studies in the management area have also studied code reuse in the open source community. My study differs from previous studies and contributes in two aspects.

First, instead of trying to identify what has been used, I focus on how innovative ideas, in the form of source code, have been diffused among open source developers. Unlike most prior studies, which start at a large scale level and identify certain reused

code from multiple projects, I start from the source code in one single file and trace how it has been modified and used through its spreading out.

Second, in order to facilitate my research, I also propose a novel research method utilizing software and techniques widely used in genetics. Traditionally, file comparison tools such as diff or FileMerge for micro level, source code within one file or in-house developed tools for macro level, and files within or across projects are often used to detect changes between files. However, the phylogenetic analysis that I propose fits better with my research question. Prior tools are mainly designed for the purpose of version control; thus they normally can only compare one file with another file (few can compare with two other files). Such tools also emphasize finding changed content, whereas in my research, I emphasize the degree of changes more than the content. Thanks to computational tools used in genetic studies, my method allows comparison of source codes from tens of thousands of files at once and analyzes the differences between the two files. It also allows investigation of the timeline of changes and creates phylogenetic trees to identify evolutionary patterns. This essay not only provides new understanding of the evolution of digital artifacts but also contributes to the methodology used for conducting big data analysis on source code.

CHAPTER 4 ESSAY THREE - METHOD-BASED VERSUS SOFTWARE-BASED DESIGN INNOVATION: A SEQUENCE-ANALYTIC SIMULATION

Introduction

Contemporary design practices are increasingly being supported by digitalized tools (Buchanan 2001). This is a broad trend which can be seen in design contexts such as software development (MacCormack et al. 2006), architectural design (Baxter et al. 2010), and chip design. Increased digitalization lends cognitive support to human designers and makes it possible to approach and solve ever more complex design problems. However, in order to cope with increased complexity, organizations often introduce newer, more complex digitalized tools. Therefore, we find ourselves in an upward spiral of digitalization, where more complex problems necessitate more complex tools in order to deal with the complexity that is introduced by both design artifacts and design tools.

Design is a fundamental activity that successful organizations must perform (Baldwin et al. 2000; Simon 1996). The extraordinary successes of innovative organizations such as Apple remind us how critical design is in today's economy. However, just as design is becoming increasingly important, it is also becoming increasingly complex. As products and services become more complicated, digitalized, and globalized, organizations looking to innovate with their design processes often need to redesign existing design processes (Sanchez et al. 1996; Yoo et al. 2006). Furthermore, the pace at which organizations need to change their designs is accelerating. In order to

deal with these challenges, organizations continue to look for ways to innovate their design routines.

There are two broad approaches to the innovation of design routines: *method-based* and *software-based.* The method-based approach to design innovation is based on structured prescriptive methodologies. Structured design methods are intended to bring consistency and more robust design capability throughout the design processes. In many fields, design methodologies are long established, such as in the architecture, engineering, and construction fields, where decades of standard practices are professionally dictated. Information systems development remains a notable exception, with a rich tradition of pluralistic methodological approaches (Hirschheim et al. 1995). Other fields, however, have experienced changes in recent years and are opening up to potential innovation with respect to design routines.

Software-based approaches, on the other hand, involve the application of integrated design and coordination software systems. Firms now routinely use advanced software such as visualization and simulation tools in designing and testing new design ideas, but they also use process-oriented design coordination and data management software that addresses issues such as configuration management, version and revision control, and electronic approvals. Software vendors embed what they describe as "best practices" within the software design environment, which may be more or less aligned with any particular methodology. The introduction of software for design coordination is often accompanied by rather radical changes in design routines as organizations discover radically different ways to configure their design routines. For example, through the implementation of advanced computer-aided design and configuration management tools,

highly innovative contexts from aerospace design (Argyres 1999) to architecture (Boland et al. 2007) have experienced radical change in both the process and product of their design routines.

Although both approaches are important, in reality, many organizations emphasize one over another. As of now there has been no research comparing these emphases. Therefore, there remain a number of unanswered questions such as when one is preferable to the other in terms of emphasis. Further, what is the shape of design activity in method-based versus software-based design innovation? Are the routines similar or different? In what way?

In this study, I explore the relative efficacy of different approaches to design routines using a simulation method rooted in empirical data from a major semiconductor manufacturer (ChipCo). ChipCo has been practicing both approaches across its global design teams for years. Through a simulation, I investigate how method-based and software-based approaches perform differently when a firm faces different levels of environmental volatility. Further, I look to characterize differences between method-based and software-based process structures.

The essay is organized as follows: next, I briefly review the distinction between different approaches to design innovation, followed by a description of event-sequencing methodology. Then I present the case description and the simulation results. I conclude with a discussion, limitations, and future opportunities.

Literature Review

Digitalization of Design Processes

There are several discourses which are relevant to the digitalization of design processes, such as routines, digitalization, and arguments concerning distributed knowledge, cognition, and agency. The literature on routines has recognized that the participation of artifacts in shaping how routines unfold is important (D'Adderio 2008). In this conceptualization, routines become inscribed in artifacts, and artifacts subsequently impact the unfolding of routines. This is reminiscent of a larger trend within IS scholarship that bases itself on the mutual intertwinement of the material and the social (Robey et al. 2003).

When such artifacts are digitalized, however, their interpretative flexibility increases drastically, and they can be recombined and reconfigured in multiple ways (Yoo et al. 2010a). Hence, their role as part of a design process becomes more ambiguous and potentially leads to larger degrees of complexity in the design process itself. Indeed, it has been suggested that digitalization changes how design processes unfold (Gaskin et al. 2014), even if it remains unclear exactly what those impacts are.

As digitalized design processes are increasingly seen as unfolding in specific patterns across time that are driven by the human designers and artifacts involved in a process, it becomes increasingly natural to recognize the ways in which agency is shared across human and material actors (Leonardi et al. 2010). While such intimations of shared agency seem to mirror arguments about the shared nature of cognition (Boland et al. 1994), shared knowledge (Tsoukas 1996), and shared practice (Cetina et al. 2005), we

still have much to learn about how the increase of complexity due to digitalization is managed in a distributed fashion.

Design Innovation: Methods vs. Software

Methodologies and software form the bases of two critical approaches in design innovation. A method-based design innovation involves the implementation of a structured sequence of activities and associated business rules and conventions in order to improve design process outcomes. The importance of methodology in design has been long studied in many literatures, particularly in software engineering and information system literature. In this literature, early structured methods such as waterfall (Royce 1970) and the system development lifecycle model (SDLC, see (Davis 1974)) were developed in order to enforce discipline and order on earlier unstructured efforts that resulted in "spaghetti code" and a large proportion of failure (Boehm 1988). The software development literature has nearly a half-century tradition of proposing and testing methods for improvement of this software development (e.g., waterfall, SDLC, prototyping, sociotechnical, RAD, spiral, agile), but the software field is not alone. Method-based approaches abound in other innovative domains as well, including product design and development (Eppinger et al. 1995), innovation identification and management (Cooper 1990), integrated circuit design (Browy et al. 1997), and architectural design (Gray and Hughes 2001) among others.

A software-based design innovation comprises the other broad approach to the improvement of design processes. Although they may be more or less consistent with particular methodologies, software-based approaches de-emphasize any particular

method and instead enable a variety of capabilities for designers to develop and coordinate their activity. For example, in the context of computer-aided software engineering (CASE) tools, certain systems (such as IBM's Rational platform) align with a particular methodology (e.g., the Rational Unified Process, or RUP, see (Kruchten et al. 2000)), whereas others may be entirely unrelated to a particular methodology (such as Microsoft's Visual Studio). In software engineering and information system literature, the role and power of CASE tools have drawn a lot of attention over the years. Many studies provide evidence that CASE tools can be associated with improved productivity (Banker et al. 1991; Norman et al. 1989). Orlikowski (1993) emphasized the organizational change needed for the adoption of such CASE tools. Software-based design environments abound in other areas as well (although they are not nearly so thoroughly studied as CASE tools). These software-based design systems include product lifecycle management (PLM) systems for product design and development (Grieves 2005); innovation management systems (Dooley et al. 2003), electronic design automation (EDA) software for integrated circuits (MacMillen et al. 2000), and building information modeling (BIM) environments for architectural design (Azhar et al. 2008).

Although there is some research into how to better integrate methods with software (Tolvanen et al. 1993), there is little research into comparing these two approaches. Given the rapid pace of innovation, however, it would make sense to understand which approach might be more beneficial for organizations in rapidly changing contexts and those in relatively stable contexts. Environmental volatility, which describes the level of uncertainty in its many dimensions in a particular organizational context, is one of the classic contingency variables in organizational scholarship

(Downey et al. 1975; Milliken 1987) and is an appropriate moderator in a studies of design innovation. In this research I look to explicitly compare these approaches in two ways. First, I look to understand the difference of their impact on performance given different levels of environmental volatility. Second, I look at the structure of the different approaches to identify how they might compare given different levels of environmental volatility.

Event Sequencing as a Way of Understanding Design Routines

In an organization, a routine can be considered as a series of closely related activities in order to accomplish certain objectives. Pentland and Feldman (2005) argued design routines as unit of analysis in the sense that design routine can be considered as "generative systems with internal structures and dynamics." The internal structures and dynamics represent an organization's DNA. Once a routine is executed, feedback on the performance will be received by the organization. In response to the feedback, the organization then will adjust how the routine will be executed next time, and the internal structures and dynamics will also be adjusted accordingly.

Gaskin et al. (2014) propose an organizational genetics approach in order to better understand design routine. Following this approach, an organizational routine can be seen as a sequence of events, which can be further decomposed into different generative elements with a set of limited values. This approach provided a way to open up the black box of an event and allow deeper understanding of organizational routine. Drawing on this approach, I conceptualize a design routine that consists of a series of activities, each

of which in turn requires using a set of affordances of tools in order to accomplish design goals.

Design Automation in Semiconductor Industry

Since the 1960s, the semiconductor industry has become an important driver of technological and economic change that enables numerous sectors, from agriculture to healthcare. Automation enabled by the digitalization afforded by embedding chips into products has significantly changed the way people think and work, as well the way firms operate and organize their activities (Yoo 2010). As the enabler of such digitally-enabled automation, the semiconductor industry is of central importance in driving the changing logics of product design.

Previous management researchers interested in work automation mainly treated the semiconductor industry as one out of many typical high-tech industries. For example, Hart and Estrin (1991) found that the interdependence of firms was changed by computer integration and inter-organization networks. Rampalli (1991) examined performance improvement by a computer-aided manufacturing system. Uzsoy, Lee and Martin-Vega (1992) studied automation in the production planning phase. Macher, Mowery and Simcoe (2002) also examined the Internet-based e-business application's impact on the semiconductor industry value chain.

However, automation in the semiconductor industry is adopted not only to facilitate routine processes such as manufacturing or operations, as is the case in many other industries, but to improve the productivity of designers. Still, the gap (as shown in

Figure 18) between design complexity and designers' productivity continues to grow as the complexity growth rate is far greater than the productivity growth rate.

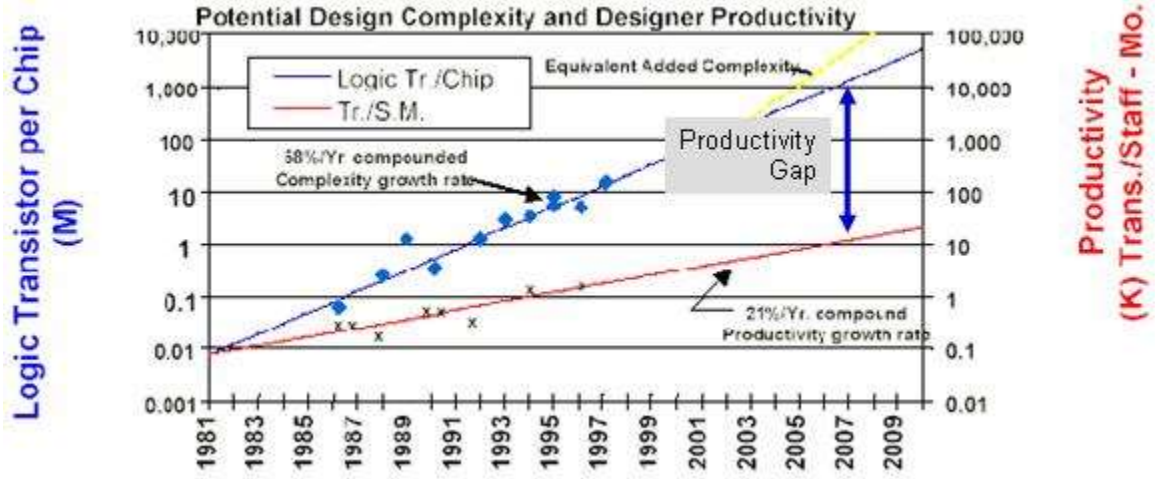

Figure 18. Gap between Design Complexity and Design Productivity9.

SEMATECH, a not-for-profit consortium for semiconductor manufacturing, considered design tools as a productivity enhancer but explained that design automation is "in no way keeping up with design complexity and may constrain the rate of density innovation … software productivity is not advancing as fast as semiconductor productivity … a slowdown in the productivity of software may ultimately constrain the semiconductor industry" (Goodall et al. 2002). Facing such a unique challenge, semiconductor companies put vast efforts into improving design automation. Yet, very few studies (if any) in management have analyzed the changes and impacts brought by design automation. Since the productivity gap was first described in the early 1980s, many tools and methodologies intended to address the gap have been invented. For example, Thomas and his colleagues (1983) proposed automatic data path synthesis,

[9] http://punetech.com/punechips-editors-blog-systemverilog-and-designer-productivity/

which is still a major automation technique used in chip design. However, the requisite variety implicated by design complexity is simply not present in contemporary chip design processes, therefore the question of how to raise productivity to match increasing design complexity is increasingly problematic and therefore of practical importance.

Integrated circuit (IC) design, like software design, has a set of well-established design methodologies to follow. A typical design flow is very like the waterfall design in software. The general design starts with the system specification, then architectural design, functional design, logic design, physical design, physical verification and manufacturing. Among all these design phases, physical design has significantly affected by the introduction of design automation techniques. Physical design is the phase during which a physical designer transforms logic/functional requirements into actual physical layout. Logic design transfers the design requirement/specification from natural description into different levels of logic abstraction, such as register transfer level (RTL) description. RTL description specifies the flow of digital signals including input and output of the digital circuits. Then the physical design phase implements specification in RTL into actual design of circuits' components like logic gates. The main objective of physical design is to place the right gates at the right place with the right wiring connecting them based on the RTL description within the constraints of time, power, and noise.

Methodology-Oriented Design vs. Software-Oriented Design

Case of ChipCo

ChipCo is a global leading semiconductor manufacturer, considered one of the most innovative companies in the world. It has been doubling the capability of semiconductors every 18-24 months for decades. To achieve such a record, ChipCo has maintained a huge design department consisting of multiple design teams across the world. Among ChipCo's global design teams, two design approaches are commonly used: structured digital design (SDD) and the physical synthesis (PS) approach.

Design Process at ChipCo

Though ideally chip design should follow the waterfall model in which physical design takes place after logic design is fixed, in most cases, a logic design team cannot fully predict the outcome and feasibility of physical design, thus logic designers have to constantly revise logic description based on feedback from physical designers. In the meantime, the change of logic will also impact the physical design. As the logic design outcome becomes increasingly complex as the result of the increased performance requirement of chips, a physical designer may find it hard to design physical placement directly from the logic description, therefore, automation design emerged as a way to help physical designers to accomplish design task in a timely manner. A design automation (DA) team designs tools to automate/facilitate the design work. Meanwhile, logic design also started to move downstream so that logic design outcomes can be directly input into the physical design tools. Such a move is accomplished by the

emergence of RTL design, which acts as an intermediary process between high level logic design and traditional physical design by abstracting high level logic description into RTL description, which can be then used by design tools directly.

Because of the ever changing nature of logic design and tied design schedule, when a new project starts, the designs from all three domains (logic, tool and physical) have to start more or less simultaneously rather than sequentially. Because chip design in ChipCo follows the Tick-Tock model, a new design project in most situations is based on previous projects with incremental changes on either performance or efficiency. Logic design starts by inheriting old logic model (base logic model) from previous project, and assumes tool and physical design will behave the same as in previous project. Tool design starts by revising the old tools to comply with the base logic model and assumes physical design will behave the same as in previous projects. Physical design starts with using old tools and the base logic model to find out any issue that may cause undesired results and provide feedback to the logic team and tool team. To sum up, in order to shorten the overall design cycle, three design teams all start at the same time based on each's own assumption about the other two teams. Afterwards, each design team concurrently works on a feedback loop until the design moves to the manufacture phase.

Three Major Design Domains: Logic Design, Physical Design and Tool Design

The core chip design activity (Table 24) has been split and distributed two times along with the evolution of design. The first split happened when logic design was separated as a result of the evolution of circuits toward digital integrated circuits. Such movement separates desired circuit behavior from physical placement: logic design

mainly deals with the functionality that a specific circuit needs to perform by abstracting design logic into machine readable models using RTL description. In other words, through logic design, the actual chip design is divided into two parts: functionality design is accomplished by logic designers, while placement design is handled by physical designers. This separation allows physical designers to focus on design problems of the routing circuit alone without considering the actual functionality required. By using the RTL model, the logic design team also digitalizes the design problem and provides the possibility of using digital tools to process the design problem. The second split happened when digital tools started automating and replacing physical designers' design activities. Thanks to the advance of technology, design automation tools can now perform various design activities from simple tasks like digitalizing a hand-drawing to very complex tasks such as synopsis RTL model into placement directly, which greatly increases physical designer's productivity.

Table 24. Different Designers Involved in Chip Design

Role	Responsibility	Outcome
Logic Designer	Translate logic requirement into RTL description, which can be understood and used by design automation tools	RTL description
Physical Designer	Use design tools to design actual physical placement based on RTL description	Physical placement
Tool Designer	Design tools that can take RTL description as input correctly and generate the desired result	Design automation tools

Two Different Design Approaches: SDD and PS

Whichever way that chip design work is structured, it is a highly complex activity. While there are a wide range of organizational and individual approaches to chip design and many overlaps between them, at ChipCo design is mainly driven by two approaches, the more traditional computer aided design approach called Structured Digital Design (SDD) and a more recent highly automated design approach called Physical Synthesis (PS).

In the SDD approach, the designer retains a higher degree of personal control over the design by selectively employing digital tools to conduct various tasks throughout the design process. Hence, high-level design expertise is often needed to make a series of critical design decisions. Therefore, with an SDD approach, the outcome of microprocessor design processes is highly dependent on the personal expertise of the chip designer driving the process. Consequently, such design processes are fundamentally manual and creative and incrementally design the chip with the help of a set of computer-assisted tools. SDD designers use their detailed knowledge of the design space and its constraints while searching for plausible solutions, which are then generated and confirmed by computer-based tools. As such, design work is carried out in an art-crafting way, it is mainly used in the area where holds the most critical data paths and thus requires the most attention. Such crafting design also makes the SDD approach less efficient in terms of design time needed per synthesized gate.

Further, the physical design can be seen as a two-phase activity. In the first phase, a schematic based on the RTL input specifications is generated. These RTL specifications determine the basic functionalities of the chip and provide a basis upon which physical designers will work to design a chip that is as fast and cheap as possible. This process is highly automated with the help of automation tools. During this transformation phase, the initial schematic will be automatically generated by the design tools, then the designer will virtualize the schematic through the computer-assisted tool and revise it by using a creative manual process. Hence, most of the design actually occurs in a simulated environment. The schematic generated during this phase provides a representation of a circuit matching the logic requirements, which will be used later as a map for physical placement. In the second phase, designers generate specific layout designs detailing the placement of gates and wiring based on the schematic. This phase is mainly concerned with executing the specifications laid out in the RTL so as to deliver the fastest chip possible at the lowest cost possible. Unlike in first phase where a SDD designer can work upon a tool-generated initial basis, now the designer relies mostly on manual work to create the layout plane via the help of digital tools.

In the more recently developed PS approach, chip designers adopt highly integrated automation techniques that can significantly improve task performance. This is done through leveraging highly complex tools that in part replace human design judgment with computer-based algorithmic design capabilities. Hence, the PS approach achieves design goals by a singular step based on complex sets of input parameters specified by the designer. This is followed by an automated solution search process. This way, several design steps that are separate in more manual design processes such as SDD

are integrated within the functionality of a single suite of design tools. Specifically, during the PS process, the design is not separated into schematic design and layout design as in the SDD approach. Instead, the design automation tools integrate these two stages such that the tool will take RTL in then directly generate placement out. Therefore, the make-up of tools is of central importance, and design outcomes will depend on the quality of inputs as well as the power and speed of tools. In this approach, a larger portion of physical design work has been replaced/carried out by these automated design tools.

One thing that must be pointed out is that although PS is supposed to automate the whole placement design process, in reality, PS designers still need to switch to hands-on tuning processes near the end of the project, when the automatically generated placement layout is very close to the desired performance, in order to minimize the potential risk brought by the tool.

A Method-based Design at ChipCo

The case of ChipCo, where two different design approaches have been used, is not unique. The SDD design in ChipCo is an example of a method-based design approach. For an SDD designer, the placement of hundreds of thousands of electronic gates/cells requires substantial cognitive effort in order to find out the right place while meeting the requirements. Because of the way SDD designers use the tool, such cognitive effort can be barely reduced by the design tools. The creative magic still resides in the human designer; the SDD designer is aware of the entire placement progression step by step and the tool is only used to help the designer execute mind work. Therefore, it

emphasizes the methodology part of design and the improvement mostly comes from a human being's better and deeper understanding of the underlying logic of the design object. The software used in this approach usually is for implementing and realizing the methodology. Human intervention therefore is the major driving force of the performance improvement of this type of design. The software used in this design approach is often modified upon request from the designers. For example, an architect draws sketches using a computer, then the software generates specified 3D models. Regarding design routine, methodology-oriented design does not require frequent change, and firms are often more conservative about organizational change. Thus, the design routine normally is more stable over time.

Software-based Design at ChipCo

PS in ChipCo is a typical software-driven design approach. As the capability of software and artificial intelligence has been significantly improved in the last decades, the role of software has also gradually shifted from assisting to (partly) replacing human judgment and implementation. More and more firms are adopting heavily tool-based design approaches in order to achieve better and faster design result. Because autonomous design tools can now generate placement directly from the RTL description, PS designers rarely need to work on routing placement directly. In fact, most design work of RLS designers essentially becomes searching and revising constraints and inputs of design tools. The cognitive effort to find the right placement, which is required of an SDP designer, has been completely replaced by RLS tools. The designers only work on the input and output of the design automation process and ignore the mechanisms of the

process. Since the actual placement design is shifted from human designer to software tools, a dedicated software development team is often deployed to improve the software capability. Human intervention becomes less critical in improving design performance. In this case, designers often change their design practice in response to the change of software. Therefore, the rapid advance of software usually requires significant changes of design routine accordingly. For example, the typical iteration cycle for one SDD-based team I interviewed at ChipCo has reduced from one week to two days in no more than one year due to a significant improvement in the software capability. However, compared with the methodology-oriented approach, this approach is typically associated with a much larger amount of investment for higher computational capability and better algorithms.

Learning in Different Environments

Due to the significant difference in the roles of human designers and software tools between SDD design and PS design, how the design routine may be improved from the feedback of design performance is expected to vary as well. That is, how designers can learn from past experience and adopt their learning in the future will be different. For a method-based design, like SDD design, the design performance highly depends on the designer's personal expertise. The knowledge required for SDD design is know-how. The designer needs to know how the circuits/electronic gates work and how certain changes may lead to certain outcomes. Though the basic electronic properties can be taught, the actual design knowledge is more on the tacit end of knowledge continuum. There are many tricks that cannot be easily articulated and transfer to other designers and require

experience to learn. Therefore, the learning in method-based design is often incremental and gradual.

In contrast, software-based design, like PS design, heavily relies on the software to carry out the actual design process. The autonomy of the tools makes the designer more like a machine operator. The knowledge required for this type of design is know-that. The designer needs to know what parameters are available to adjust the software instead of how the internal logic works in the software. This knowledge is more on the explicit end of the knowledge continuum, where the operation of software can be more easily taught by others. Much of the learning from the feedback is then shifted from the designer to the software vendor. The software vendor can have centralized, systematic, and organized learning which can be implemented into the software and adopted by the designer. The effort needed to adopt the software update is generally much less than the personal learning required in method-based design, especially when the software update only alters the design algorithm instead of the interface. Because changing the algorithm has little impact on the use of the software, the greater the changes from learning to be implemented, the greater the difference between the efforts needed for the two approaches.

In reality, firms may face different environments, which can greatly affect the performance of design process. Here, I use the concept of environmental volatility to identify two types of environments a firm may face: low volatile environments and high volatile environments. Snyder and Glueck (1982) suggest that environmental volatility can be further decomposed into two parts: market volatility and technological volatility. Since I am interested in the performances of different design approaches, I mainly focus

on technological volatility. Thus, in my case, environmental volatility mainly refers to R&D competition in an industry. Because design performance is measured as how well a design goal is achieved, the effect of environmental volatility on design performance does not come from the design process, rather it mainly comes from the effect on the design objective affected by different environmental volatility. I propose two hypotheses for how environmental volatility will affect the performances of two design approaches.

In a low volatile environment, a firm has less pressure from its competitors and is more tolerant of slower, incremental improvement or even failure. Thus, the performance requirement is to some extent lowered by such an environment. In this situation, designers are willing to try out different ways to solve problems without worrying too much about failure. This allows designers to learn more from new design routines regardless of performance change and find better solutions. A method-based designer will benefit more in such an environment as the effort needed is usually more than for designers using software-based design. A method-based designer is more likely to achieve the same degree of change as a software-based designer. Therefore, I propose:
HYPOTHESIS 1: In a low volatile environment, a method-based approach to design innovation and a software-based approach to design innovation will yield similar performance improvement over time.

In contrast, high environmental volatility features more pressure from competitors and is less tolerant of slow improvement. Thus, the need to improve performance is increased by the environment. In this case, designers will not have much chance to try out different ways because failure often means losing to competitors and thus cannot be accepted. Such a scenario hurts the method-based designer much more than the software-

based designer. The firm then falls into a vicious circle: fewer changes lead to fewer opportunities, then fewer opportunities lead to fewer chances to improve. However, for software-based design, because the changes to improve require much effort from the designer, it is less stressful for them to implement changes during the design. More changes brought by advances in software provide more opportunities for designers to find better solutions. Thus I propose:

HYPOTHESIS 2: In a high volatile environment, a software-based approach to design innovation will yield better performance improvement over time than a method-based approach.

A Two-Stage Simulation Analysis

Model Setup

In this study, I set up a two-stage simulation model based on data collected from ChipCo. I have collected data from seven projects cross different design phases from my field studies. Based on my interviews with five different designers, I created seven design routines (Figure 19) using the notations, all of which have been verified by those designers in terms of their accuracy and completeness. Then I calculated the transition matrixes which are used in my simulation model as the initial seed. A detailed explanation of the matrixes and how they have been used is given in the following part. In general, the first step involves simulation of the design routine using the Markov Chain method and the second step involves simulation of the organization's adjustment of the balance between exploration and exploitation. In my simulation, I set each simulation run to generate 50 generations of routines, which means an organization will

adjust its design routine 49 times. Considering one design routine may last from several weeks to several months in my empirical data, I believe 50 routines covers a long enough period to discover patterns of performance change. Moreover, to overcome the randomness of the MCMC method, I performed 500 simulation runs and aggregated all results.

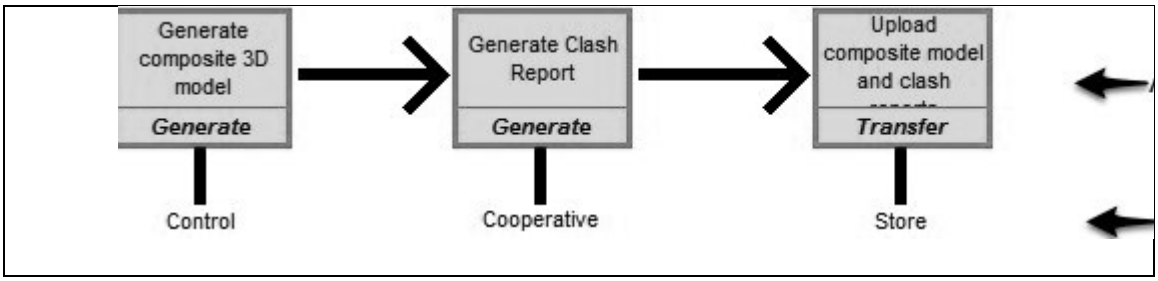

Figure 19. Example Design Routine at ChipCo.

Data Collection for Initial Setting

In this study, I collected data through following the design teams at ChipCo through interviews over the course of three years. During this period of time, two projects (A and B) were covered. I had interviews with designers from logic design, tool design team and physical design teams as well as the managerial domain. I primarily focused on designers from the physical design team, where digitalization/design automation had the most significant impact on design activity. Table 25 shows the summary of interviews I did. The interviews were conducted in a semi-structured manner that gave interviewees more freedom to describe their design experience related to the topic of interest. The interviews were conducted during field visits roughly every half year, corresponding to the design progress. Each design project was generally finished in about two years.

Table 25. People Interviewed Distinguished by Project and Work Process

	PS	SDD	Management	Tools	Total
Project A	4	3	7	2	16
Project B	17	4	13	3	37
Total	21	7	20	5	53

After the interviews were conducted, I followed Gaskin et al.'s (2014) organizational genetics approach to create the design routines for each project and verified them with the designers to make sure the routines represented the actual design process.

Stage One: Two-Level Markov Simulation of Design Routine

Following Gaskin et al.'s (2014) organizational genetics approach, I propose a simplified model in my study. In this model, I decompose an event into two elements: activity and affordance. At the higher level, a design routine consists of a series of activities; at a lower level, each design routine consists of a set of affordances. Therefore, each activity defines the objective and boundary of one event, representing the core of the event. Consistent with Gaskin et al. (2014), I define eight types of activities (see Table 26). I also include affordance, which has five different values (Table 27) in my model. Affordance is a concept originally developed by ecological psychologist Gibson (1979) and has been increasingly used in the information systems literature (DeSanctis et al. 2003; Markus 2005; Norman 1999) to understand the relationship between practice and artifact. It describes possible actions associated with the activity and artifact. It helps us to distinguish activities of the same type, as same-type activities may actually have

different numbers and/or types of affordances. In addition, cost and time of each event can be more precisely measured with additional information offered by affordance.

Table 26. Summary of Activities

Pseudonym	Activity Type	Description
1	Start	Dummy type defining the start of a design routine
2	Generate	Activity that generates new idea or object
3	Choose	Activity to select one from multiple options
4	Negotiate	Activity to communicate and negotiate with other people
5	Execute	Activity to perform or execute a plan or procedure
6	Validation	Activity to validate
7	Transfer	Activity to transfer object from one place to another place
8	Complete	Dummy type defining the end of a design routine

Table 27. Summary of Affordances

Pseudonym	Affordance Type	Description
1	Representation	Functionality that enables the user to define, describe or change a definition or description of an object, relationship or process
2	Analysis	Functionality that enables the user to explore, simulate, or evaluate alternate representations or models of objects, relationships or processes
3	Transformation	Functionality that executes a significant planning or design task, thereby replacing or substituting for a human designer/planner
4	Control	Functionality that enables the user to plan for and enforce rules, policies or priorities that will govern or restrict the activities of team members during the planning or design process
5	Cooperative Functionality	Functionality that enables the user to exchange information with another individual(s) for the purpose of influencing (affecting) the concept, process or product of the planning/design team

I developed a two-level (activity-affordance) simulation model using Matlab based on the MC method. The methodology of this model is shown in Figure 20.

Transition between activities (step 3 and 6) is based on the Markov transition matrix of activities, which is generated from my empirical data. Transition between activities stops after a "Complete" activity is generated. "Transition" between affordances (steps 2 and 5) is based on a Markov affiliation matrix of affordance.

Unlike activity, which is temporally sequential, affordances are not sequential in my model. The affiliation matrix is used to represent the coexisting relationships between certain affordances. "Transition" between affordances stops when the maximum number of affordances is generated. To link the two levels together (steps 1 and 4), the first affordance is generated based on the frequency distribution functions for each activity type. Also, the maximum number of affordances is generated based on the affordance count distribution of each activity type. To sum up, I have one activity transition matrix, six affordance affiliation matrices, six affordance frequency distributions, and six affordance count distributions. Tables 28 and 29 show example simulated results.

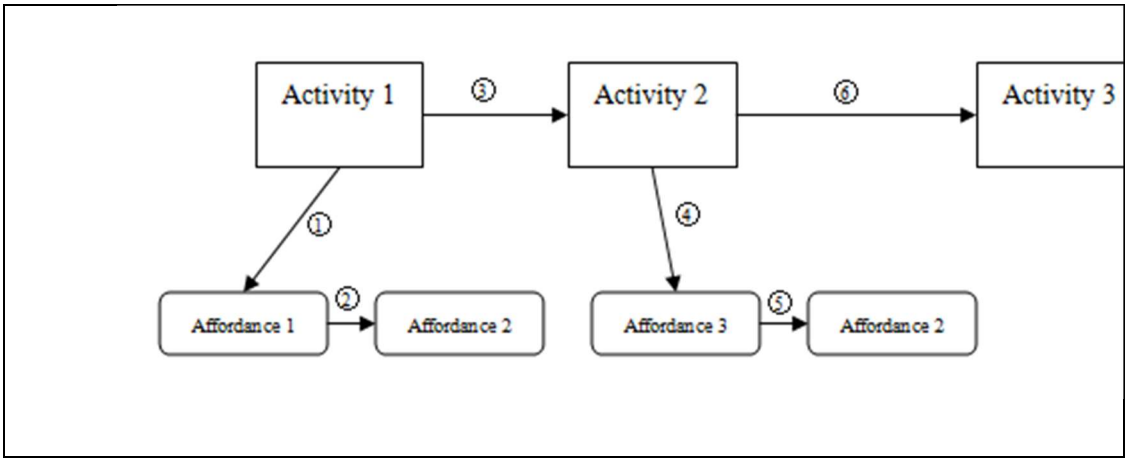

Figure 20. Two-Level Simulation Model.

Table 28. Activity Level Simulation Result

Routine #	Activity Sequences
1	1 3 2 4 7 6 4 2 6 5 6 4 6 5 6 4 6 5 6 4 6 4 5 8
2	1 2 2 2 4 6 3 2 5 7 6 4 6 6 4 4 4 6 4 4 6 4 5 6 4 6 6 4 6 4 6 4 4 8
3	1 4 4 5 6 5 7 2 4 6 6 4 6 3 8

Table 29. Affordance Level Simulation Result

Activities	Affordances
1	1 5
3	2 4 3 4 1
2	2 5 3 1 1
4	3 5
7	5 5 2 3 1
6	1 1 2
4	5 4 1 5
2	5 5
6	1
5	4
6	5 5
4	3 5 3 4
6	5
5	2 2 3
6	2 1 5 1
4	3 5 3 2
6	3 2 2 5
5	1 1 1 3
6	3 4 4
4	4 3 4
5	2
8	3 2

Stage Two: Simulation of Organizational Adjustment

After one routine has been generated, performance of this routine can be calculated and used to determine how much an organization will learn from this routine. If the current routine yields a better performance than expected, then an organization is likely to learn more and vice versa. In the simulation model, activity transition matrix is updated through Equation 1 below.

$$M = M + (a + \Delta P)M'; \alpha \in [0,1], \Delta P \in (-\infty, +\infty) \quad \text{(Equation 1)}$$

Parameter α represents an organization's design approach; a large α suggests a more software-based approach to design innovation has been adopted, which also indicates a more aggressive view that more changes in design routine are preferred. ΔP represents the performance change over expected performance. Positive ΔP suggests a performance improvement. M and M' are Markov matrices of the old routines and current routine, respectively representing organization's existing knowledge and possible new knowledge. When holding ΔP constant, M will be more influenced by M' as α increases, which means that for the same performance change, an organization with a software-based approach is more willing to adopt the routine change in the future. When holding α constant, new M will be more influenced by M' as ΔP increases, which means that the organization will learn more from new practice if the routine provides better performance. In the model, the matrix is updated in the form of a transition frequency

(count of each type transition) matrix and then converted to a transition possibility matrix. Tables 30 and 31 show the Matlab results of the matrix updating process.

Table 30. Initial Transition Matrix

0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.2139	0.2539	0.1083	0.1731	0.1099	0.0049	0.4577	0.0000
0.3048	0.0098	0.0637	0.0049	0.0275	0.0346	0.0229	0.0000
0.2674	0.2900	0.3822	0.1406	0.1648	0.5849	0.0915	0.0000
0.0000	0.1367	0.1274	0.0422	0.1786	0.1584	0.0687	0.0000
0.0535	0.1631	0.1274	0.4806	0.4547	0.1925	0.2677	0.0000
0.1604	0.0781	0.0637	0.1295	0.0275	0.0148	0.0687	0.0000
0.0000	0.0684	0.1274	0.0291	0.0371	0.0099	0.0229	0.0000

Table 31. Example Transition Matrix after 99 Updates

0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1783	0.2551	0.0307	0.1915	0.0645	0.0052	0.4951	0.0000
0.4295	0.0214	0.0913	0.0059	0.0441	0.0626	0.0939	0.0000
0.1976	0.2091	0.5138	0.1189	0.1823	0.4906	0.0916	0.0000
0.0000	0.2205	0.0053	0.0655	0.1078	0.1616	0.0722	0.0000
0.0459	0.1283	0.0730	0.3866	0.4275	0.1408	0.1916	0.0000
0.1488	0.0625	0.0094	0.1121	0.0054	0.0469	0.0087	0.0000
0.0000	0.1031	0.2766	0.1195	0.1792	0.0924	0.0468	0.0000

Evaluating the Performance

Measuring the performance of a design practice can be complex. In the traditional project management literature, cost, scope and schedule are the three most important

factors in measuring this performace. A project is considered successful when an expected objective is met within time and cost constraints. As I define one event with activity and affordance, I first assign a weight to each affordance type, representing both the time and cost associated with that affordance, so that different activities of the same type are actually differently represented in my model in terms of cost and time. I then assume all routines' objectives as one standard unit of objective. For example, project A with an objective which is 2 times more complex than project B's objective can be considered as project A', with the same objective as project B but requiring one-half the cost and time. In other words, what has been measured here in fact is the efficiency of completing a same level objective. In the simulation, a higher requirement of design routine then can be translated to requirement of smaller sum of affordance weights. I then can compute the performance P of a routine as

$$\sum_{i=1}^{5} W_i * Aff_i$$

where W is the weight for certain affordance and Aff is the total number of that affordance in a routine.

To further refine the measurement, I then define a valid range of design performance. I believe only within a certain range can the routine be considered as successfully finished. This helps to eliminate unrealistically high and low results due to the randomness of MCMC.

Last, I compute the performance change as in Equation 2 to get the ratio of the current routine's performance difference to the expected performance.

$$\Delta P = (E(P) - P) / E(P) \qquad \text{(Equation 2)}$$

Environmental Effect

As I discussed before, environmental volatility may affect performance over time differently for the two approaches. To incorporate such environmental effect, I introduce an environment parameter *b* into my model, modifying the expected performance so that $E(P) = b\overline{P}$, where $\overline{P}$ is the average performance of past routines. Parameter *b* is the environmental effect on the expected performance. When the environment is highly volatile because competitors are rapidly improving their performance, a firm will try to improve at the industry pace rather than at its own pace, so *b* is set less than 1. In this environment, intense competition among firms requires a company to set a higher standard each time in order to keep up with expected improvement from its competitors. In contrast, when *b* is larger than 1, it suggests a low volatile environment. With less competition, firms are more tolerant of slow and/or minor performance improvement. Taking all this into account, I get Equation 3.

$$M = M + (a + ((b\overline{P} - P) / b\overline{P}))M'; a \in [0,1], b \in [0,1] \qquad \text{(Equation 3)}$$

Results

The results from the simulation show us how organizations with different approaches to design innovations – specifically software-based versus method-based – will evolve over time. Before I go into detail, I first run two-way factorial ANOVA test to check if the approaches, environmental volatility, and their interaction have a significant effect on performance. As shown in Figure 21, I do find significant effects from these factors and their interactions.

	Number of obs =	200	R-squared	=	0.4697
	Root MSE =	10.9657	Adj R-squared =		0.4616

Source	Partial SS	df	MS	F	Prob > F
Model	20879.197	3	6959.73234	57.88	0.0000
Approach	1817.39382	1	1817.39382	15.11	0.0001
Volatile	14823.8027	1	14823.8027	123.28	0.0000
Approach#Volatile	4238.00053	1	4238.00053	35.24	0.0000
Residual	23568.3262	196	120.246562		
Total	44447.5232	199	223.354388		

Figure 21. ANOVA Result for Performance Differences in High Volatility.

Figures 22 and 23 show the results when *b* is set to be 0.8, which represents a high volatile environment; the performance differences over time are clear. A software-based approach to design innovation outperformed a method-based approach by a large margin. A method-based design innovation approach did not provide significant performance improvement; the performance only changed from the 80s range to the 70s range. By contrast, the software-based approach improved performance to the 40-50 range. The one-way ANOVA results also provide evidence that approach type does significantly affect firm's performance over time. Thus H1 is supported.

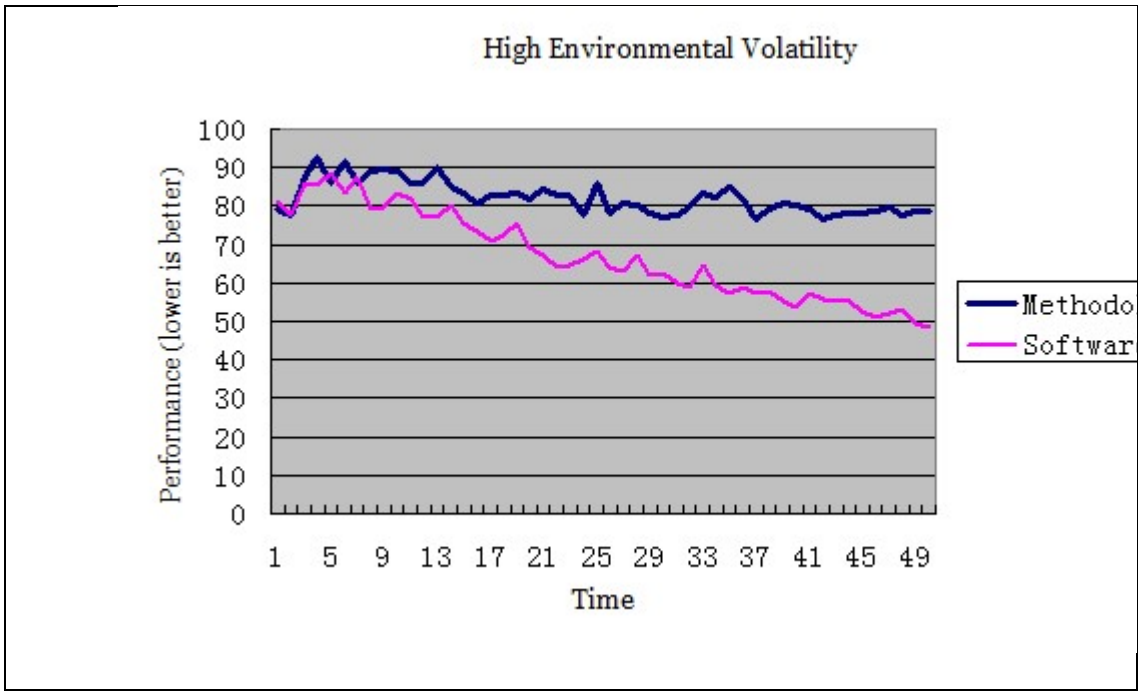

Figure 22. Performance Differences (lower is better) in High Volatile Environment.

	Number of obs =	100	R-squared	=	0.4389
	Root MSE =	8.70117	Adj R-squared	=	0.4331

Source	Partial SS	df	MS	F	Prob > F
Model	5802.9658	1	5802.9658	76.65	0.0000
Approach	5802.9658	1	5802.9658	76.65	0.0000
Residual	7419.61932	98	75.7104012		
Total	13222.5851	99	133.561466		

Figure 23. ANOVA Result for Performance Differences in High Volatility.

In Figures 24 and 25, *b* is set to be 1.2, which represents a low volatile environment. Here, there are not significant performance differences between a software-based approach and a method-based approach. Both performances changed from 80s to

30s, and the change follows a similar pattern as a software-based approach performance in a high volatile environment. ANOVA test results also show there is no significant effect of approach type. H2 is also supported by the simulation result.

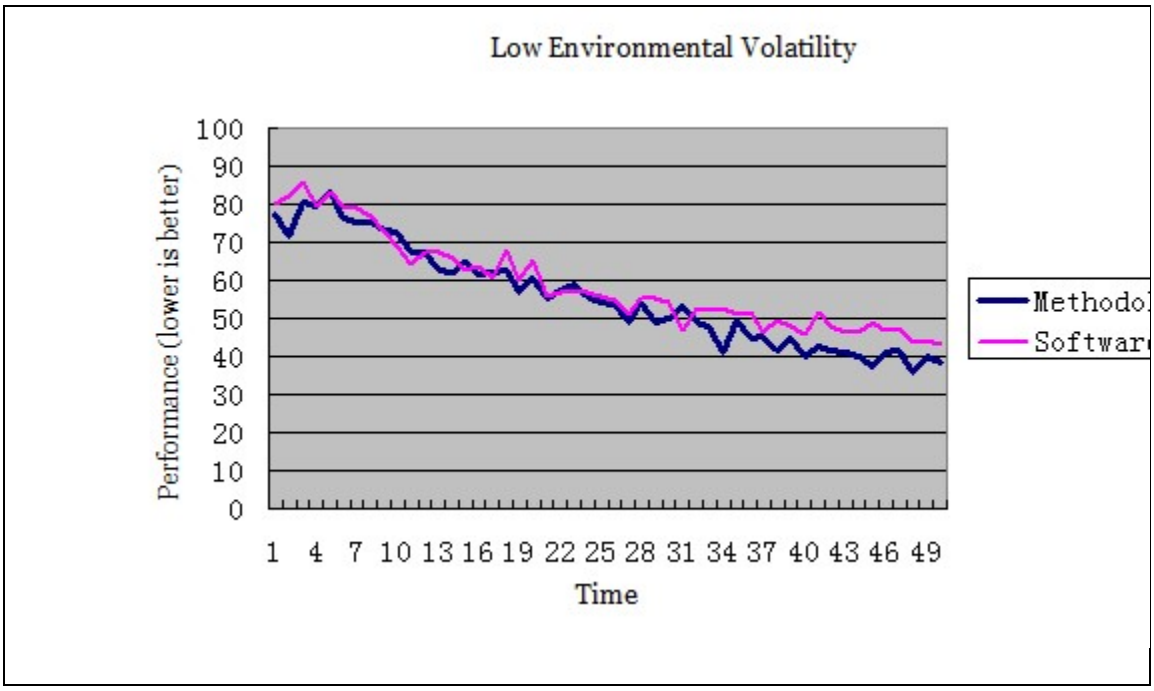

Figure 24. Performance Differences (lower is better) in Low Volatile Environment.

	Number of obs =	100	R-squared	=	0.0154
	Root MSE =	12.8368	Adj R-squared	=	0.0053

Source	Partial SS	df	MS	F	Prob > F
Model	252.428544	1	252.428544	1.53	0.2188
Approach	252.428544	1	252.428544	1.53	0.2188
Residual	16148.7069	98	164.782723		
Total	16401.1354	99	165.668034		

Figure 25. ANOVA Result for Performance Differences in Low Volatility.

Understanding the Evolutionary process of Design

In addition to the performance difference over time, I am also interested why such performance difference may occur. I follow an organizational genetics approach to examine the change in design routines. The key to this approach is a cross-level analysis with a focus on the complex interactions between the system and its underlying "genetic" elements. This approach will also help to understand the complexity of the *phenotype* (i.e., the distinguishing characteristic of living organisms) via its *genotype* (i.e., the basic genetic building blocks of the artifact). In my case, the phenotype is the performance, while the genotype is the sequence of activities. This is the same rationale by which geneticists study species via understanding their DNA. The phylogenetic analysis takes two steps. The first step is to align the sequences of the final routines. I only use the sequence of activities in the alignment. The reasons to exclude affordance are: first, the matrices of affordances do not update during evolution in my current model, thus I do not expect much impact from the affordance level; second, this allows us to focus more on the overall change of routine at a high level. Then in the second step, following a similar approach to the one I developed in Essay 2, I was able to produce phylogenetic trees for each scenario based on the design routines generated from the simulation.

By looking at the trees generated by the four different cases (Figure 26 and Figure 27), I find that the average length of a tree for a method-based approach in a highly volatile environment is much bigger than that of the other three situations: its sum of branch length is 18.45, compared with 13.03, 13.54 and 13.03 for the other trees. This suggests that the design routines for this situation (a method-based approach in a high

volatility environment) become more divergent than in other situations. It seems to suggest that there may be a best practice (design routine) out there. However, the divergence shows that the firm with a method-based approach will have a hard time to find the best practice leads to the best performance. The other trees are more convergent.

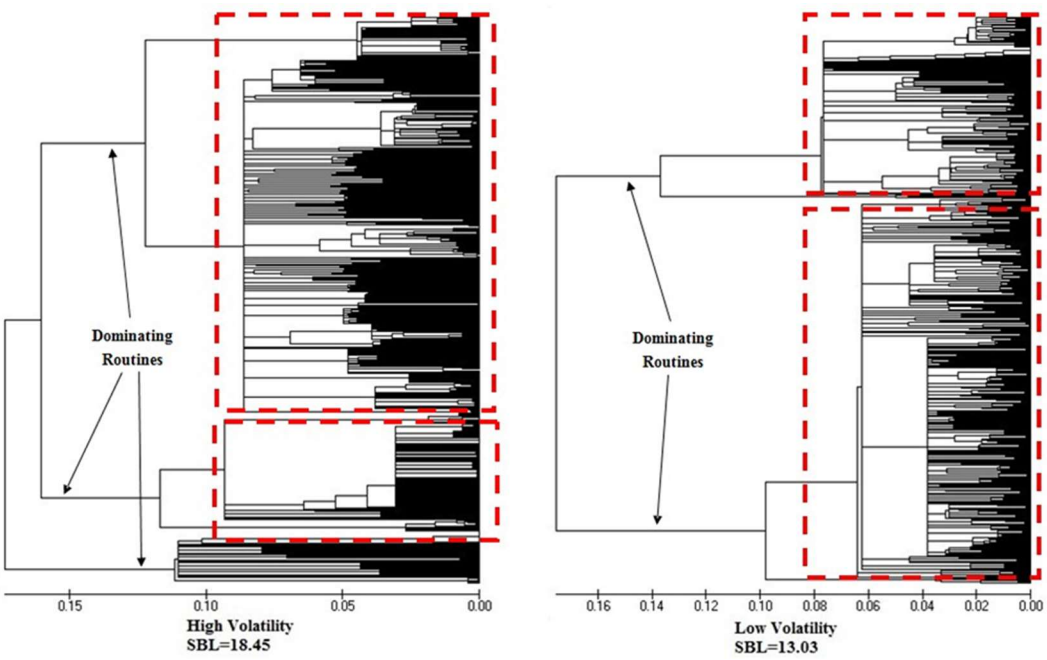

Figure 26. Phylogenetic Trees of Method-based Approach in Different Environments.

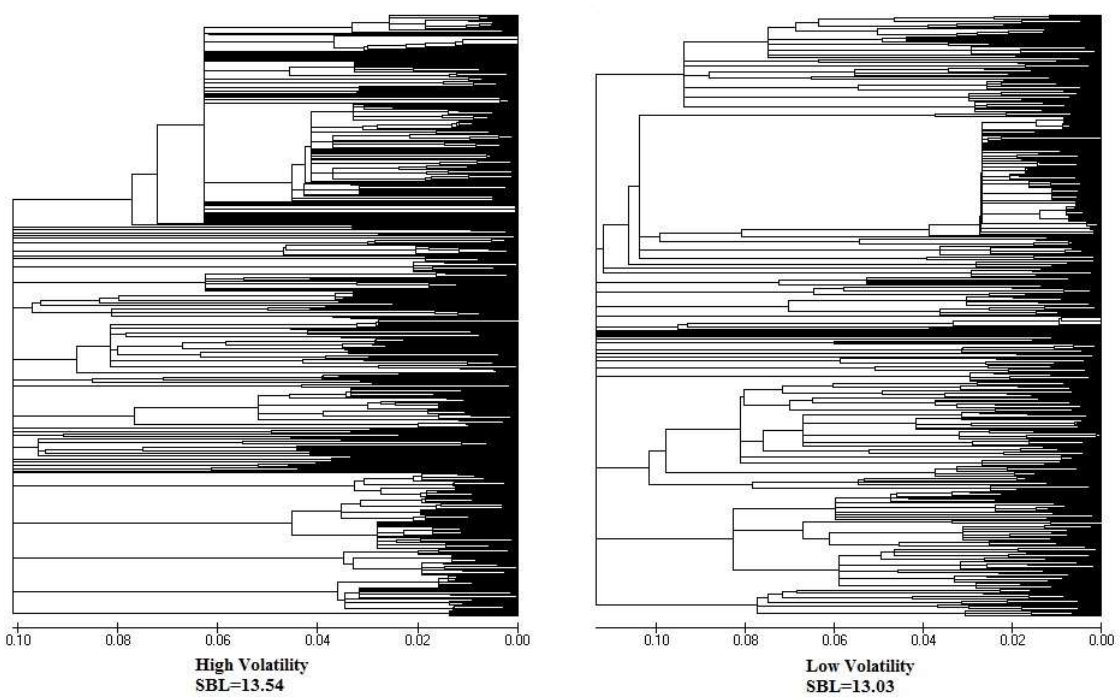

Figure 27. Phylogenetic Trees of Software-based Approach in Different Environments.

By comparing the evolutionary patterns (topologies of tree), I clearly see the differences between a method-based approach and a software-based approach. I find more balanced patterns in Figure 26 and highly punctuated patterns in Figure 27. This means, for a method-based approach, the diversity of design routines mainly derives from a few dominating routines (long branches at left of tree), where clusters emerge. This does not contradict my findings about divergence and actually shows different patterns of divergence. For a software-based approach, I do not see such major clusters. The divergence starts from the very beginning. In other words, for a method-based approach, the routines tend to converge into a few clusters while still diverging more in terms of difference. For a software-based approach, routines do not converge into clusters, but

overall they do not diverge as much as routines of the method-based approach. This difference implies that a method-based approach focuses more on exploiting old certainty, while a software-based approach emphasizes exploring new possibilities.

Discussion

In this study, I investigate if and how two approaches to design innovation, namely method-based and software-based, will perform differently under different competitive conditions. Our simulation analyses provide a rather clear result. When the environment is volatile, organizations that pursue design innovations by deploying powerful software that brings radical changes in design routines achieve much stronger performance than those pursuing method-based approaches. By contrast, when the environment is static, there were no significant differences between the two approaches. Given that the introduction of new software often involves high costs including software, training, opportunity cost, as well as various implementation risks, one might argue that organizations will be better served by opting for a method-based innovation approach in a static environment. In other words, the benefit of software-based design is not without cost. The practical implication from my study is that due to the high cost of IT investment, adopting a software-based approach to design innovation may not benefit a firm that is not facing intense competition. Software-based approaches are best adopted by firms in a highly competitive market.

The phylogenetic analysis in my study also provides additional insight to explain the performance differences. I show that method-based approaches produce much more divergent design routines in a volatile environment than the other three situations. The reason for such high divergence is most likely due to the slow search process. In my

simulation model, the design routine is improved via updating the transition matrix of different activities. Three factors determine the degree of updating. The updating mechanism follows $M = M + (a + ((b\overline{P} - P) / b\overline{P}))M'$. When *a* and *b* are both small, the new matrix is much less influential. In other words, the firm has a much lower rate of learning from new practice, even when the new practice gives a better performance. Only when performance is significantly better than expected performance will the firm really learn. Because a firm with a method-based approach has fewer opportunities to change design routines, it is less likely to benefit from performance improvement due to changes in routines. This is especially serious in a volatile environment since there is less chance of getting significant performance improvement. In a static environment, the chance of getting significant performance improvement is much higher, so the disadvantage of the method-based approach is less severe.

This finding provides potential implications for management at firms using method-based approaches to improve their performance in a volatile environment. The key is to encourage designers to be open to new opportunities, to increase *a*. The large *a* in the software-based approach is the result of the radical improvement of software capability that forces the organizational routine to change accordingly. However, there are also many other ways to increase *a*, such as an open-minded atmosphere, a corporate culture more tolerant of failure, and so on. In science, many important findings and successes are actually the result of failure. The invention of Post-It Notes by 3M is a classic example.

From the organizational learning perspective, the method-based approach and the software-based approach in fact represent two learning mechanisms: exploitation and exploration. As can be seen from the different evolutionary patterns in Figures 26 and 27, on one hand, various design routines of the method-based approach mostly derive from a few major routines, suggesting this approach attempts to exploit the full potential of a few possibilities. On the other hand, the tool-based approach tends to learn from exploring numerous possibilities with fewer constraints, as no dominant routine can be identified in the phylogenetic trees. The different evolutionary patterns of these two approaches highlight the importance of diversity in digital evolution, which cannot be easily revealed using conventional approaches. The lack of dominating routines during the evolution process in software-based design practice leads to far more different kinds of routine branches, which increase the likelihood of design routines with better performance.

Limitations and Future Opportunities

This study is not without limitations, but these may lead to future research opportunities. First, the simulation is based on my empirical data collected from ChipCo, a high-technology company famous for its digital innovation over decades. Whether my findings can be generalized to all other industries needs further investigation. Semiconductor design has always been at the frontier of technological innovation, so I will not be surprised if my results do not hold in some other industries. However, given the significance of the semiconductor industry in leading technological change, my study is still very valuable to the related managers or administration. By conducting similar

analysis in other industries, future researchers can examine what factors may cause different evolution patterns across different industries.

Second, in my current model, I set all types of affordances to have the same constant weight due to a lack of accurate evidence for different weights. Better and more accuate assessment of affordances associated cost and time can be extremely difficult. Without sufficient and reliable data to support different weights, I set all weights the same for the sake of simplicity. Future researchers should differentiate the impacts of different affordances on overall performance and generate more accurate results. In addition, my model assumes no new affordances can be introduced. However, it is possible that new affordances can show up with the introduction of new technology. Since all affordances are basically equal in my current model, this problem is less serious. However, this should be carefully considered in future studies of situations in which affordances are not equal.

Lastly, with regard to the application of phylogenetic analysis in social science study, it is similar to my earlier studies that there is also a violation of the assumption of independence of mutation taking place in each individual activity. The Markov activity transition matrix activity, however, can inform the correlation among activities, thus making it possible to adjust the violation. Future researchers can improve my result by incorporating the interdependence of mutations into different activities.

Conclusion

In this study, I try to understand the different performances of two major design innovation approaches: a method-based approach versus a software-based approach. I ground my research in empirical works with industry partners. Based on the empirical

data, I build a two-stage simulation model to forecast the evolution of design routines in different situations. My findings provide new insight on the effect of design approach in different environments and the reason behind such different effects. I also offer practitioner recommendations for choosing an approach choice in light of different competition environments and suggest on other possible ways to improve performance without altering design approaches.

CHAPTER 5 SUMMARY AND CONCLUSION

Digital technology has shaped our world dramatically, both technically and socially. In this thesis, I propose an evolutionary view to examine the digital artifacts, particularly micro-level mutation as the evolution mechanism. I demonstrate the application and benefit of this evolutionary view in three different contexts. In this chapter, I first summarize the overall contribution of my thesis and then the specific implications of each essay. I conclude with limitations and future research opportunities.

Contributions

The evolutionary view I proposed in this thesis is a form of computational social science inquiry that combines both a theoretical framework and a computational analytics-supported methodology that can be used to examine the changing process of digital artifacts. Following the path of evolutionary thinking like evolutionary economists like Nilson and Winter and evolutionary ecologists like Hanna and Freeman, I also apply ideas from Darwinian evolution to social science. However, the big difference and primary contribution of my study is that I emphasize different evolution mechanisms. Instead of natural selection, I argue that many changes in digital artifacts or other similar contexts like technology innovation can be seen as similar to biological mutation, which is another important yet less known evolution mechanism that creates variations and makes selection possible. This also makes applying modern biological analysis possible in the social science context and complements prior studies that consider natural selection as the primary evolution mechanism by recognizing the importance of mutation. My work also does not assume the superiority of changes in digital artifact. I emphasize the

diversity stimulated by digital evolution and study how to identify the evolutionary patterns that lead to different kinds of diversity; offers theoretical foundations for future studies.

In addition to academic contributions, my work also provides many insights to designers, managers, and policy makers. My approach can potentially map the relationships among seemingly unrelated artifacts and create a tree of digital life. With the exponential growth of digital artifacts, being able to have a complete picture like biologists have of all kinds of species is rewarding by itself. Better understanding of the past can help better predict the future. By looking at how newer digital artifacts are related to older ones, digital artifact designers can better position their future designs to increase their chances for success. For managers and policy makers, it is critical to predict what digital artifacts are evolving into in order to better guide and regulate the development, such as preventing a would-be wrongful path or promoting a beneficial path.

In addition, this evolutionary view also emphasizes the importance of the evolutionary process. It complements the conventional co-variance or network-based analyses by providing additional insights from the analysis of evolutionary process in two particular ways: first, it helps to bridge the dependent variable to the independent variables by answering *how* changes in independent variables are causing changes in the dependent variable. For example, in the third essay on digital design, it is not hard to demonstrate the relationship between the use of different design approaches and the design performance. However, how the design process of different design approaches is actually evolving in different competitive environments can only be captured through this

evolutionary view in order to see how the design routine is gradually changing from the first generation to the last generation and eventually leading to different performance.

Secondly, my evolutionary view of digital evolution illustrates the temporal process, the evolutionary process in my case, as an alternative to traditionally widely-used structural relationship or variable attributes in explaining the changes. Like prior scholars (Abbott 1990; Gaskin et al. 2014; Pentland and Feldman 2007) who showed the importance of action-in-sequence which can be used as a powerful way to understand the social dynamic, I also argue that change-in-sequence, the evolutionary process, can be a powerful way to understand digital artifacts. For example, in the first and second essays, four evolutionary patterns can be identified based on the two evolutionary properties of rate and direction. These patterns can then be used as process variables in a similar way as attribute variables or structural variables. In addition, this approach is developed along with the increasing availability of digital trace data which is able to capture nuanced changes in digital evolution. It offers a new way to utilize the large volume of trace data beyond simply aggregating and treating like a single data point.

Lastly, by analyzing the temporal changes of digital evolution, I am able to identify different evolutionary patterns of digital artifacts through the two dimensions, evolutionary direction and evolutionary rate. Those two dimensions represent the characteristics of evolutionary patterns and can be then used with conventional analysis approaches to help uncover other possible hidden relationships, such as how different evolutionary directions may lead to different survival rates of firms.

The first essay makes three key contributions. First, it develops a novel analytical approach to quantitatively measure digital evolution, which provides a way to better

understand digital evolution and a foundation to test a series of hypotheses. Through phylogenetic analysis, it maps the different mashups based on their predicted ancestral relationship. Second, it expands current knowledge about technological evolution by introducing two important properties, evolutionary direction and evolutionary rate, based on the identified phylogenetic tree. Four types of evolutionary patterns can be identified by using the two evolutionary properties. Third, it provides explanations for the formation of different evolutionary patterns in different environments and offers web API providers with new insights on ways to promote use of web API.

The second essay contributes to existing research on open source development by putting the research in a much broader social networking context that applies the concepts of change in traditional collaboration (much bigger scale of collaboration) and change of integration in social networking (more ways to communicate) to the open source community. Leveraging the computational analytics developed in the first essay, it also emphasizes the development process and provides an evolutionary framework to better understand OSS development through its development process. It also uses evolutionary rate and evolutionary direction to classify four different development patterns based on the revisions done to the project. By showing the relationship between different evolutionary patterns and different levels of competition, Essay 2 provides insight to coding platform providers such as Github or Sourceforge about how they can make their platforms more vital as well as advice to companies such as Twitter or Facebook using these platform for open innovation about how they can make their projects more popular and active.

The last essay examines the design performance improvement as the result of evolution of two types of design practice, method-based design and software-based design, under different circumstances of competition. It provides evidence of benefits brought by fast evolution of digital design in a highly competitive environment. The phylogenetic analysis of simulated design evolution reveals the process of design routine as it is changed from the original to the end, which leads to certain performance improvement. The existence of dominant design routines echoes the debate about the trade-off between exploration and exploitation. My result therefore offers practical implications to firm management about the choice of a particular design approach when facing different competitive environments.

Limitations

My thesis left several important limitations as future research opportunities. First, the violation of assumptions of the biological evolution model, including the interdependence among changes at different places and the different mutation possibilities, warrants additional work. This type of limitation is very common when applying natural science methods to social science studies. The tools used in natural science are often not optimized for handling data from social science because of the dynamics and interdependent nature of the social world and human activity. This requires closer cooperation between natural scientists, social scientists, and computer scientists working together to improve the tool set. Violation of the assumptions of natural science may create different results in the social context. With improved tools, social scientists should in the future reexamine and compare the results and investigate where the difference comes from. Secondly, the changes in social context, like mashup

development or design routine, are usually much larger in degree compared with changes in phenotype in biology. For example, one alteration of a basic element in a mashup or design routine may represent a large percentage of change, compared with biology, where the total change of a whole DNA sequence can sometimes be less than 0.1%. For example, human beings and chimps share 95% common genes[10]. Therefore, the degree of change in digital evolution is far larger than in biology. Such a difference could create potential issues and the issues are still unknown. As far as I know, a biologist studying a virus faces somewhat similar problems as I described in that the change or mutation scale may be far larger than normal mutation. Future researchers can benefit from learning about virus evolution patterns.

Another limitation of my research is the generalizability of my work into other fields. My essays rely on a particular type of subjects, such as jQuery plugin projects or semiconductor design firms. This is largely a decision based on the availability of data and computational complexity. However, it is still very important to repeat similar analyses in different contexts and to test the validity of my approach.

Future Opportunities

There are several other future research opportunities. First, due to the scope of this thesis, I only analyzed the past evolutionary pattern. Ultimately, we'd like to understand the antecedents and consequences of all four evolutionary patterns. What are the factors that affect evolutionary direction and evolutionary rate? Aside from environmental pressure and competition, are there any other factors that influence digital

[10] http://genetics.thetech.org/ask/ask404

evolution, such as communication among participants? Future researchers can also examine the future pattern of evolution. Does digital evolution always follow its previous patterns? Is there any turning point or threshold at which one type of evolutionary pattern may change to a different one?

Secondly, future researchers can also try to integrate my work with evolutionary economics or population ecology and simultaneously examine the roles of mutation and natural selection in creating the various forms of organization or digital artifacts and how the two mechanisms can interact. Mutation and selection are considered two arms of evolution that together cause the variety of species. Similarly, we should also examine how minor changes in lower-level social activities can be reflected in higher-level social dynamics and ultimately affect social changes under environment pressure.

Last but not least, how can the other two evolution mechanisms, genetic drift and migration, be applied in social science? This topic is extremely interesting as the neutral theory of molecular evolution has already devalued the role of natural selection in evolution and genetic drift is now considered the primary force. What is the equivalent of genetic drift in social changes? Will we be able to discover similar phenomena in social changes?

Overall, my thesis is the first attempt to conduct computational social science from an evolutionary view. It focuses on the evolutionary process of digital artifacts as a way to inquire about the complex social world and highlights the importance of mutation. Although my current analyses are still largely limited by the computational capability to deal with the complexity of an extremely large volume of digital trace data and the violations of some biologic assumptions, they open a new door on the “micro” nature of

digital evolution. Building upon the evolutionary view proposed in my thesis, future researchers will discover a lot more novel insights.

REFERENCES

Anderson, P., and Tushman, M. L. 1990. "Technological discontinuities and dominant design: A cyclical model of technological change," *Administrative Science Quarterly* (35), pp 604-633.

Andersson, B. A., and Rade, I. 2001. *Material constraints on technology evolution: the case of scarce metals and emerging energy technologies*, (Citeseer.

Argyres, N. S. 1999. "The impact of information technology on coordination: Evidence from the B-2 "Stealth" bomber," *Organization Science* (10) 1999, pp 162-180.

Arthur, W. B. 2009. *The Nature of Technology: What It Is and How It Evolves*, (Reprint edition ed.) Free Press.

Azhar, S., Hein, M., and Sketo, B. 2008. "Building Information Modeling (BIM): Benefits," *Risks and Challenges*) 2008.

Baldwin, C. Y., and Clark, K. B. 2000. *Design Rules, Vol. 1: The Power of Modularity*, (First Edition edition ed.) The MIT Press: Cambridge, Mass.

Banker, R. D., and Kauffman, R. J. 1991. "Reuse and productivity in integrated computer-aided software engineering: an empirical study," *MIS quarterly*) 1991, pp 375-401.

Baxter, R. J., and Berente, N. 2010. "The process of embedding new information technology artifacts into innovative design practices," *Information and Organization* (20) 2010, pp 133-155.

Benner, M. J., and Tushman, M. L. 2002. "Exploitation, exploration, and process management: The productivity dilemma revised," *Academy of Management Review* (28:2), pp 238-256.

Bijker, W. E. 1995. *Of Bicycle, Bakelites, and Bulbs: Toward a Theory of Sociotechical Change*, (The MIT Press: Cambridge, MA.

Boehm, B. W. 1988. "A spiral model of software development and enhancement," *Computer* (21) 1988, pp 61-72.

Boland, R. J., Lyytinen, K., and Yoo, Y. 2007. "Wakes of innovation in project networks: the case of digital 3-D representations in architecture, engineering, and construction," *Organization Science* (18:4) 2007, pp 631-647.

Boland, R. J., Tenkasi, R. V., and Te'eni, D. 1994. "Designing information technology to support distributed cognition," *Organization Science* (5:3) 1994, pp 456-475.

Bortolussi, N., Durand, E., Blum, M., and François, O. 2006. "apTreeshape: statistical analysis of phylogenetic tree shape," *Bioinformatics* (22) 02/01/2006, pp 363-364.

Boudreau, K. 2010. "Open platform strategies and innovation: Granting access vs. devolving control," *Management Science* (56) 2010, pp 1849-1872.

Bowler, P. J. 1992. *The eclipse of Darwinism: Anti-Darwinian evolution theories in the decades around 1900*, (JHU Press.

Browy, C., Gullikson, G., and Indovina, M. 1997. "A Top-Down approach to IC design," *Integrated Circuit Design Methodology Guide*) 1997.

Brynjolfsson, E., and McAfee, A. 2014. *The Second Machine Age*, (New York.

Buchanan, R. 2001. "Design research and the new learning," *Design issues* (17) 2001, pp 3-23.

Bush, A., Tiwana, A., and Rai, A. 2010. "Complementarities Between Product Design Modularity and IT Infrastructure Flexibility in IT-Enabled Supply Chains," *IEEE Transactions on Engineering Management* (57) May 2010, pp 240-254.

Callon, M., Latour, B., and Rip, A. (eds.) *Mapping the Dynamics of Science and Technology*. Macmillan, Basingstoke, 1986.

Campbell, D. T. 1965. "Variation and selective retention in socio-cultural evolution," *Social change in developing areas: A reinterpretation of evolutionary theory* (19) 1965, pp 26-27.

Cetina, K. K., Schatzki, T. R., and von Savigny, E. 2005. *The practice turn in contemporary theory*, (Routledge.

Chesson, P. L., and Huntly, N. Year. "Community consequences of life-history traits in a variable environment.," 1988, pp. 5-16.

Churchill Jr, G. A. 1979. "A paradigm for developing better measures of marketing constructs," *Journal of marketing research*) 1979, pp 64-73.

Clark, K. B. 1985. "The interaction of design hierarchies and market concepts in technological evolution," *Research policy* (14) 1985, pp 235-251.

Cooper, R. G. 1990. "Stage-gate systems: a new tool for managing new products," *Business horizons* (33) 1990, pp 44-54.

D'Adderio, L. 2008. "The performativity of routines: Theorising the influence of artefacts and distributed agencies on routines dynamics," *Research Policy* (37) 2008, pp 769-789.

Davies, T. J., Cooper, N., Diniz-Filho, J. A. F., Thomas, G. H., and Meiri, S. 2012. "Using phylogenetic trees to test for character displacement: a model and an example from a desert mammal community," *Ecology* (93) February 28, 2012, pp S44-S51.

Davis, G. B. 1974. *Management Information Systems: Conceptual Foundations*, (

Dereeper, A., Guignon, V., Blanc, G., Audic, S., Buffet, S., Chevenet, F., Dufayard, J.-F., Guindon, S., Lefort, V., Lescot, M., Claverie, J.-M., and Gascuel, O. 2008. "Phylogeny.fr: robust phylogenetic analysis for the non-specialist," *Nucleic Acids Research* (36) Jul 1, 2008, pp W465-469.

DeSanctis, G., Fayard, A.-L., Roach, M., and Jiang, L. 2003. "Learning in online forums," *European Management Journal* (21) 2003, pp 565-577.

Dooley, L., and O'Sullivan, D. 2003. "Developing a software infrastructure to support systemic innovation through effective management," *technovation* (23) 2003, pp 689-704.

Downey, H. K., Hellriegel, D., and Slocum Jr, J. W. 1975. "Environmental uncertainty: The construct and its application," *Administrative science quarterly*) 1975, pp 613-629.

Emmanuelides, A. P. Year. "DETERMINANTS OF PRODUCT DEVELOPMENT TIME: A FRAMEWORK FOR ANALYSIS.," Academy of Management1991, pp. 342-346.

Eppinger, S. D., and Ulrich, K. T. 1995. "Product design and development," *Product design and development*) 1995.

Ethiraj, S. K., and Levinthal, D. 2004. "Modularity and innovation in complex systems," *Management Science* (50) 2004, pp 159-173.

Faraj, S., Jarvenpaa, S., and Majchrzak, A. 2011. "Knowledge Collaboration in Online Communities," *Organisation Science* (22:5), pp 1224-1239.

Faulkner, P., and Runde, J. 2010. "The Social, the material, and the ontology of non-material technological objects," *Documento de trabajo*) 2010.

Floyd, I. R., Jones, M. C., Rathi, D., and Twidale, M. B. Year. "Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software," IEEE2007, pp. 86-86.

Gaskin, J., Berente, N., Lyytinen, K., and Yoo, Y. 2014. "Toward a Generalizable Sociomaterial Inquiry: Tools and Techniques for Analyzing Patterns of Association in Sociomaterial Routines," *MIS Quarterly* (38) 2014, pp 849-871.

Gawer, A. 2011. *Platforms, markets and innovation*, (Edward Elgar Publishing.

Gibson, J. J. 1979. "The ecological approach to visual perception.,") 1979.

Goodall, R., Fandel, D., Allan, A., Landler, P., and Huff, H. R. Year. "Long term productivity mechanisms of the semiconductor industry," 2002.

Grant, P. R. 1972. "Convergent and divergent character displacement," *Biological Journal of the Linnean Society* (4) 1972, pp 39-68.

Grieves, M. 2005. *Product Lifecycle Management: Driving the Next Generation of Lean Thinking*, (McGraw Hill Professional.

Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., and Tatham, R. L. 2006. *Multivariate data analysis*, (Pearson Prentice Hall Upper Saddle River, NJ.

Hannan, M. T., and Freeman, J. 1977. "The population ecology of organizations," *American journal of sociology*) 1977, pp 929-964.

Hargadon, A. B. 2002. "Brokering knowledge: Linking learning and innovation," *Research in Organizational behavior* (24) 2002, pp 41-85.

Hart, P., and Estrin, D. 1991. "Inter-organization networks, computer integration, and shifts in interdependence: the case of the semiconductor industry," *ACM Trans. Inf. Syst.* (9:4) 1991, pp 370-398.

Hirschheim, R., Klein, H. K., and Lyytinen, K. 1995. *Information systems development and data modeling: conceptual and philosophical foundations*, (Cambridge University Press.

Hughes, T. P. 1993. "The Evolution of Large Technological Systems," in *The social construction of technological systems: New directions in the sociology and history of technology,* W. E. Bijker, T. P. Hughes and T. J. Pinch (eds.), MIT Press: Cambridge, pp. 51-82.

Jonathan Davies, T., Meiri, S., Barraclough, T. G., and Gittleman, J. L. 2007. "Species co‐existence and character divergence across carnivores," *Ecology Letters* (10) 2007, pp 146-152.

Kallinikos, J. 2007. *The consequences of information: Institutional implications of technological change*, (Edward Elgar Publishing.

Kallinikos, J., Aaltonen, A., and Marton, A. 2010. "A theory of digital objects," *First Monday* (15) 2010.

Kashtan, N., Noor, E., and Alon, U. 2007. "Varying environments can speed up evolution," *Proceedings of the National Academy of Sciences* (104:34), pp 13711-13716.

Kauffman, S., and Macready, W. 1995. "Technological evolution and adaptive organizations: Ideas from biology may find applications in economics," *Complexity* (1) 1995, pp 26-43.

Kessler, E. H., and Chakrabarti, A. K. 1996. "Innovation speed: A conceptual model of context, antecedents, and outcomes," *Academy of Management Review* (21) 1996, pp 1143-1191.

Kruchten, P., Célier, V., and Trybocki, C. 2000. *Introduction au Rational Unified Process*, (Eyrolles.

Leonardi, P. M., and Barley, S. R. 2010. "What's under construction here? Social action, materiality, and power in constructivist studies of technology and organizing," *The Academy of Management Annals* (4) 2010, pp 1-51.

MacCormack, A., Rusnak, J., and Baldwin, C. Y. 2006. "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science* (52) 2006, pp 1015-1030.

Macher, J. T., Mowery, D. C., and Simcoe, T. S. 2002. "e-Business and disintegration of the semiconductor industry value chain," *Industry and Innovation* (9:3) 2002, pp 155-181.

MacMillen, D., Camposano, R., Hill, D., and Williams, T. W. 2000. "An industrial view of electronic design automation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* (19) 2000, pp 1428-1448.

Markus, M. L. 2005. "Technology-shaping effects of e-collaboration technologies: Bugs and features," *International Journal of e-Collaboration (IJeC)* (1) 2005, pp 1-23.

Marshall, A. 1907. "The Social Possibilities of Economic Chivalry," *The Economic Journal* (17) 1907, pp 7-29.

Mayr, E. 2001. *What evolution is*, (Basic books.

Mihaescu, R., Levy, D., and Pachter, L. 2009. "Why neighbor-joining works," *Algorithmica* (54) 2009, pp 1-24.

Milliken, F. J. 1987. "Three types of perceived uncertainty about the environment: State, effect, and response uncertainty," *Academy of Management review* (12) 1987, pp 133-143.

Norman, D. A. 1999. "Affordance, conventions, and design," *interactions* (6) 1999, pp 38-43.

Norman, R. J., and Nunamaker Jr, J. F. 1989. "CASE productivity perceptions of software engineering professionals," *Communications of the ACM* (32) 1989, pp 1102-1108.

Orlikowski, W. J. 1993. "CASE tools as organizational change: Investigating incremental and radical changes in systems development," *MIS quarterly*) 1993, pp 309-340.

Pearce, J. A. 2002. "Speed merchants," *Organizational Dynamics* (30) 2002, pp 191-205.

Pentland, B. T., and Feldman, M. S. 2005. "Organizational routines as a unit of analysis," *Industrial and corporate change* (14) 2005, pp 793-815.

Pinch, T. J., and 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), pp 399-441.

Porter, M. E. 1990. "The competitive advantage of nations," *Harvard business review* (68) 1990, pp 73-93.

Rampalli, P., Ramesh, A., and Shah, N. 1991. "CEPT-a computer-aided manufacturing application for managing equipment reliability and availability in the semiconductor industry," *Components, Hybrids, and Manufacturing Technology, IEEE Transactions on* (14) 1991, pp 499-506.

Robey, D., Schwaig, K. S., and Jin, L. 2003. "Intertwining material and virtual work," *Information and organization* (13) 2003, pp 111-129.

Royce, W. W. Year. "Managing the development of large software systems," Los Angeles1970, pp. 1-9.

Saitou, N., and Nei, M. 1987. "The neighbor-joining method: a new method for reconstructing phylogenetic trees.," *Molecular biology and evolution* (4) 1987, pp 406-425.

Sanchez, R., and Mahoney, J. T. 1996. "Modularity, flexibility, and knowledge management in product and organization design," *Strategic management journal* (17) 1996, pp 63-76.

Saviotti, P. P., and Mani, G. S. 1995. "Competition, variety and technological evolution: A replicator dynamics model," *Journal of Evolutionary Economics* (5) 1995/12/01, pp 369-392.

Simon, H. A. 1996. *The Sciences of the Artificial*, (MIT Press.

Snyder, N. H., and Glueck, W. F. 1982. "Can environmental volatility be measured objectively?," *Academy of Management Journal* (25) 1982, pp 185-192.

Stoltzfus, A., and Cable, K. 2014. "Mendelian-Mutationism: the forgotten evolutionary synthesis," *Journal of the History of Biology* (47) 2014, pp 501-546.

Thomas, D. E., Hitchcock Iii, C. Y., Kowalski, T. J., Rajan, J. V., and Walker, R. A. 1983. "Automatic data path synthesis," *IEEE Computer* (16:12) 1983, pp 59-70.

Tiwana, A., Konsynski, B., and Bush, A. A. 2010. "Research commentary-Platform evolution: Coevolution of platform architecture, governance, and environmental dynamics," *Information Systems Research* (21) 2010, pp 675-687.

Tolvanen, J.-P., and Lyytinen, K. 1993. "Flexible method adaptation in case," *Scandinavian Journal of Information Systems* (5) 1993, pp 51-77.

Tsoukas, H. 1996. "The firm as a distributed knowledge system: a constructionist approach," *Strategic management journal* (17) 1996, pp 11-25.

Tushman, M. L., and Anderson, P. 1986. "Technological discontinuties and organizational environments," *Administrative Science Quaraterly* (31), pp 439-465.

Tushman, M. L., and Murmann, J. P. 1998. "Dominant designs, technology cycles and organizational outcomes," *Research in Organizational Behavior* (20), pp 231-266.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. 1992. "A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning," *IIE transactions* (24) 1992, pp 47-60.

Veblen, T. 1898. "The Beginnings of Ownership," *American Journal of Sociology* (4) 1898, pp 352-365.

Wasserman, S., and Faust, K. 1994. *Social network analysis: Methods and applications*, (Cambridge university press.

Webb, C. O. 2000. "Exploring the phylogenetic structure of ecological communities: an example for rain forest trees," *The American Naturalist* (156) 2000, pp 145-155.

Webb, C. O., Ackerly, D. D., McPeek, M. A., and Donoghue, M. J. 2002. "Phylogenies and Community Ecology," *Annual Review of Ecology and Systematics* (33) 2002, pp 475-505.

Weiss, M., and Gangadharan, G. R. 2010. "Modeling the mashup ecosystem: structure and growth," *R&d Management* (40) 2010, pp 40-49.

Woese, C. R. 2004. "A New Biology for a New Century," *Microbiology and Molecular Biology Reviews* (68) 2004-06-01, pp 173-186.

Worden, R. P. 1995. "A speed limit for evolution," *Journal of Theoretical Biology* (176:1), pp 137-152.

Yoo, Y. 2010. "Computing in Everyday Life: A Call for Research on Experiential Computing," *Mis Quarterly* (34:2) 2010, pp 213-231.

Yoo, Y. 2012. "Digital Materiality and the Emergence of an Evolutionary Science of the Artificial," *Materiality and Organizing: Social Interaction in a Technological World*) 2012, p 134.

Yoo, Y., Boland, R. J., and Lyytinen, K. 2006. "From organization design to organization designing," *Organization Science* (17:2) 2006, pp 215-229.

Yoo, Y., Henfridsson, O., and Lyytinen, K. 2010a. "Research commentary-The new organizing logic of digital innovation: An agenda for information systems research," *Information Systems Research* (21) 2010, pp 724-735.

Yoo, Y., Lyytinen, K., Thummadi, V., and Weiss, A. Year. "Unbounded innovation with digitalization: A case of digital camera," 2010b.

Yu, S., and Woodard, C. J. Year. "Innovation in the programmable web: Characterizing the mashup ecosystem," Springer2009, pp. 136-147.

Zittrain, J. L. 2006. "The generative internet," *Harvard Law Review*) 2006, pp 1974-2040.