Strategizing Software Development: Strategic Management of Internet Service Development

Masao Kakihara
School of Business Administration, Kwansei Gakuin University
1-1-155 Uegahara, Nishinomiya, Hyogo 662-8501, JAPAN
Tel: +81 798 54 6205
Email: kakihara@kwansei.ac.jp

ABSTRACT
This paper explores a theoretical linkage between software engineering and strategic management. Software engineering is now faced with two dynamic innovation streams: technological innovation and market innovation. Harshly shook by rapid technological development and highly volatile market environments, today's software development is under the constant necessity for swift and reliable development practices and market launch in appropriate timing. In short, software development has to be more and more strategic. Based on a brief review of the existing strategic management frameworks, the paper suggests that Eisenhardt’s framework of 'Strategy as Simple Rules' is highly applicable to software development practices. Through a short case study of internet service development in Japan, the paper also suggests that dual roles of beta versions, as a product and media, would play a critical role in making strategic decisions in internet service development.

Categories and Subject Descriptors
D.2.9 [Management], K.6.3 [Software Management]

General Terms
Management, Human Factors, Theory

Keywords
Software Development, Strategic Management, Internet Service

1. INTRODUCTION
All aspects of our social lives are now greatly dependent upon various kinds of software. Ranging from common application software like spreadsheets to embedded systems installed in various kinds of electronic appliances, software has become an irreplaceably critical part of our social systems. In order to study it, the research filed of software engineering was founded in the 1960s, seeking ‘better’ development of software. So far, a number of development models and theories have been devised and supported today’s software development practices, including object-oriented programming, CASE tools, CMMI, and so on.

It is obvious, however, that software engineering is now faced with great social transformation. For software engineering, whose traditional research question has been how to technically build reliable software, there is a strong need to offer social and managerial accounts of software development practices. In short, software engineering must explain not only ‘how efficient’ but also ‘how effective’ those practices are in a given context. It is no longer a ‘closed’ research field staying within a highly technical domain but must rather be ‘open,’ incorporating with broad research achievements in social sciences such as economics, management, and sociology.

In today's turbulent business environment, software development is not an isolated practice confined only in technical fields; any software development needs to consider how to adapt to such environmental changes and to launch into an appropriate market in appropriate timing. Furthermore, managers have to consider not just whether software projects are ‘done properly’ but also whether produced software makes expected results and, in some cases, financial returns. Put it simply, today's software development has to be more and more strategic. The more dynamic and turbulent environmental changes facing software development are, the more strategic the development practices have to be to cope with those changes efficiently and effectively. However, despite such increasing importance of the strategic management perspective in software development, the software engineering research has not accommodated itself to a wide range of research achievements in business studies in general and strategic management in particular.

This paper explores a theoretical linkage between software engineering and strategic management by looking at strategic management issues of today’s software development practices, particularly the case of internet service development. Internet services, such as portals, web search services, and web-based applications, are one of the latest software businesses taking full advantage of state-of-the-art software technologies, and hence can be considered as a good example of software development of our time.

The paper is structured as follows. Section 2 describes an outline of environmental changes with which software development is now faced. Section 3 discusses some strategic management issues in the existing software engineering research. Section 4 briefly reviews research achievements of the strategic management
research and focuses on Eisenhardt’s framework of ‘Strategy as Simple Rules.’ Section 5 discusses the applicability of Eisenhardt’s framework to software development practices with a short case study of internet service development in Japan. Finally, Section 6 summarizes the paper’s analysis and arguments and discusses future work.

2. Environmental Changes and Software Innovation

Nowadays, given its ubiquity in our social lives, software has become essential to the very functioning of our society. However, such importance of software has not been sufficiently understood for a long period of time. In the dawn of the Computing Age, roughly from the 1940s to the 1950s, software was a problem domain inseparably woven into hardware issues. Through commercialization of computers in the 1960s, when epoch-making IBM’s System/360 was launched, people had gradually realized the existence of software and its important roles. In 1975, Microsoft, the largest software company in the world, was founded by Bill Gates and Paul Allen. Even at that time, few could imagine that the software company would have grown tremendously in scale and social impact as it is today. In concert with Microsoft’s gigantic success, we have seen during the last three decades the evolution of software as a research and a business field.

During the last three decades, software has experienced a wide range of environmental changes, which can be summarized as follows:

1. Rapid evolution of hardware

Hardware systems have evolved rapidly: rapid increase of CPU performance, sharp decline of price of memory and storage devices such as HDD, diversification of input/output devices, etc. Moore’s Law, the empirical observation that the complexity of an integrated circuit, with respect to minimum component cost, will double in about 18 months, seems still continue, and such rapid evolution of hardware inevitably affects the ways of developing and operating software.

2. Widespread usage of software in society

It is since the 1980s that computers have been used not only in quite limited situation such as research laboratories and governments but also in ordinary people’s social lives. Personal computers have been diffused in households and people use a variety of software in their PC. Moreover, software systems embedded in electronic appliances such as mobile phones and TV displays are required to perform stably and correctly in diverse contexts, from seas to deserts. Such widespread usage of software inevitably demands unprecedented levels of quality and reliability of software.

3. Diversification of stakeholders in software development

The scale of software development projects is ever-increasing, and this resulted in rapid diversification of stakeholders in the projects. For example, whereas the number of lines of the source code of the Windows 3.1 launched in 1992 is about 3 million, that of Windows 2000 launched in 2000 has been increased into 35-60 million [13]. In order to cope with such a rapid increase in scale and complexity of software development, the recent software engineering research has invented component-based development approaches facilitating more distributed development practices across organizational boundaries [9]. As a result, diverse stakeholders can participate in a particular software development project, and this necessitates new management approaches.

4. Continuous revision of software functions

Today’s software can and has to evolve even after its market launch. Rapid and continuous changes of software usage as described above have brought forth an increasing demand for functional revision of software systems. Diffusion of the Internet has further energized this trend, enabling post-launch distribution of program patches for update of software functions. Even package software is never completely ‘finalized,’ frequently revised by update patches through the Internet. Such never-ending revision processes create new triggers for malfunctions and makes ROI evaluation of software development projects more and more difficult.

To summarize these rapid environmental changes, software development of today must take account of two innovation streams: technological innovation and market innovation. Based on these two streams, three schematic models of software innovation can be identified (see Figure 1).

Figure 1: Three schematic models of software innovation

(1) The ‘technology-push’ model

(2) The ‘market-pull’ model

(3) The ‘interactive’ model
The first model of software innovation is one that actively takes advantage of functional capability enabled by rapid technological innovation into new software development. Here it could be called the ‘technology-push’ model. As discussed above, the pace of technological innovation is still extremely rapid. Only within a year, many technological impossibilities turn to be possible thanks to new technological innovation. Technological innovations are not just limited to hardware innovations such as increase of CPU power and network speed but can be also software innovations such as new programming language and supporting programming techniques. Strongly ‘pushed’ by these innovations, the functionalities and quality of software will be continuously innovated as well.

The second model is one that software innovation is ‘pulled’ by changing market environments such as demand levels and user preferences. This could be called the ‘market-pull’ model. In general, every software system is developed for fulfillment of certain market needs. For instance, spread-sheet applications are made to meet user’s needs of efficiency and convenience in data handling in various business scenes. Likewise, embedded software installed in mobile phones enables much higher and more complex functionalities than a simple talking function. Strongly ‘pulled’ by such market demands, software will also be innovated.

In reality, however, these two schematic models of software innovation are not discrete or antithetical but rather interrelated and interactive. Software development in reality should be conducted in dynamic interaction between technological innovation and market innovation. This is the third model which can be called the ‘interactive’ model. Today’s software development should dynamically adapt its processes and frameworks to both innovation streams, technological and market. Such a ‘dialogue’ with both changes would be a critical point of today’s software innovation.

3. Strategic Issues in Software Development

As discussed above, software development, which had been conducted in a quite ‘closed’ manner, is now in the midst of dynamic environmental changes induced by rapid technological and market innovations. In short, today’s software development has to be more strategic than ever.

Hardware performances of home-use PCs are now so improved that hardware systems rarely restrict software performances except for professional-use software such as graphic design and data simulation. Moreover, ordinary electronic appliances such as mobile phones and TVs are competing not just on hardware performances but nowadays heavily on software-based functional differences. It can be argued that software is not just a supporting system that make hardware work properly but rather a critical factor that strongly affects the quality and integrity of a product or service. Thus it is obvious that today’s software development has to include various strategic factors in its decision making processes.

The software engineering research has paid its attention to strategic issues in software development, but quite limitedly. Traditionally, formal processes of software development are structured from ‘requirements analysis’ to ‘architecture and design’ to ‘coding’ to ‘test’ to ‘deployment and maintenance.’

This process model is generally labeled the ‘waterfall’ model. Based on this linear process model, many modified models have been devised, such as ‘incremental’, ‘spiral’, ‘concurrent’, and ‘evolutional’ models [18]. However, the waterfall model is still considered as a reference model and widely used in real software development practices mainly because sequential ordering of the development phases and inhibition of backtrack are particularly convenient for project management [19].

In the software engineering research, it has been often argued that strategically important is ‘up-front’ phases in the development process, namely, ‘requirements analysis,’ and ‘architecture and design.’ These ‘up-front’ phases are the ‘human-intensive’ processes in which managerial and business intents and technological possibilities are strategically negotiated to define core features of the developed software [19]. It is also often argued that the more dynamic and ambiguous the environment in which software is launched is, the more detailed and deliberate the ‘up-front’ phases of software development should be in order to decrease operational risks in the lower phases.

This ‘Big Design Up Front’ approach, however, is now often criticized since the approach would be quite unrealistic particularly for software development in rapidly changing environments. Rapid environmental changes inevitably create unexpected changes in requirements for software in the course of development processes. Given the above discussion of software innovation models, software development processes are constantly faced with two rapid innovation streams: technological innovation and market innovation. New technological innovations can drastically change technological assumptions for software development. Likewise, new market innovations can greatly transform targets to be reached by developed software. These rapid and constant environmental changes make ‘up-front’ planning and design seriously difficult or even problematic. McConnell [12] is one of the main proponents who criticize such a ‘Big Design Up Front’ approach. He stresses the importance of ‘construction’ of software, rather than planning or design. He argues that software construction, focusing on coding and debugging, is ‘the central activity of software development’ (p. 7). Supported by this kind of argument, software engineering scholars and practitioners have created more flexible and adaptable development approaches and models such as ‘agile’ and ‘adaptive’ software development [2, 10].

Yet it can be argued that these recent software development approaches and models put ‘too much’ emphasis upon technical skills of coding and debugging and largely ignore the strategic importance of planning and design. To be sure, the ‘Big Design Up Front’ approach is in fact problematic in rapidly changing environments and it is understandable that the recent software engineering research is pursuing cultivation of feasible construction skills for adaptive development for efficient and effective software systems. However, this over-reaction against ‘Big Design Up Front’ is also problematic since it tends to result in the praise of ‘No Design Up Front.’ Such an all-or-nothing discourse would be quite unproductive for sound development of the software engineering research.

As discussed above, strategic management of software development is increasingly crucial for efficient and effective management of software development. In reality, a ‘good’ strategy for software development would be located in the realm
between two extremes: ‘excessive design’ and ‘no design.’ What software engineering in the Age of Uncertainty and Complexity needs is a strategic framework that links design, construction, and implementation together to cope with dynamic environmental changes. So far, the software engineering research poorly refers to vast research achievements of business studies in general and the strategic management field in particular. Whereas some management scholars are studying software development practices from a strategic management perspective [e.g. 3, 4, 11], research endeavor of the software engineering research looking at strategic management issues is scarce. Given that today’s software development has to cope with rapidly changing environments, this lack of reference to strategic management issues in the software engineering research will seriously harm practical validity to software development practices in the field.

4. Applying Strategic Management Frameworks to Software Development

This section offers a brief outline of recent achievements of the strategic management research and tries to apply them to software development practices.

Strategic management as a research field is one of the youngest among business and management studies. There must be little objection to the opinion that it is Porter’s study on competitive strategy [15, 16] that first systematized various strategic management issues for modern firms and started the contemporary strategic management research. Based on industrial organization economics, Porter argues that firms need to realize industry structure that determines their profitability in the industry and hence shapes their competitive strategy. He defines the elements of industry structure, widely known as ‘five forces,’ as suppliers, buyers, new entrants, substitutes, and industry competitors. Based on the analysis of industry structure, firms need, he argues, to determine their unique strategic positioning to gain competitive advantages. Rooted in Porter’s study, the so-called Positioning School is still in the main stream of strategic management research.

Responding to the flourishing Positioning School, there has been some criticism to its position-based framework of strategic management. Porter’s framework rests upon the S-C-P (Structure-Conduct-Performance) paradigm of industrial organization economics, which demonstrates that a firm’s competitive advantage is determined a priori by ‘structure,’ an outside environment that surrounds the firm. However, some scholars critically argue that a firm’s competitiveness can be also determined by unique resources held inside the firm. For example, the success of Japanese manufacturing firms, especially automobile companies, in the 1980s could not be well explained based on the position-based strategy framework [17]. Thus, firms’ resources, such as unique technological advantages and tacit organizational capability and knowledge, are gradually focused as a source of sustainable competitive advantage. Barney [1], one of the main proponents of this Resource-Based View (RBV) of the firm, proposes the VRIO framework, namely, value, rarity, imitability, and organization, to systematically analyze capabilities of the firm. The framework particularly focuses on the uniqueness of resources and organizational capability that contributes long-term competitive advantages.

Although many other strategy frameworks and approaches have been proposed so far [14], these two frameworks, position-based and resource-based, are widely accepted as the mainstream of the contemporary strategic management research. However, both frameworks are apparently questionable in applicability to software development practices. That is to say, the presupposed time-scale in strategic decision making is extremely dull. Both frameworks assume a relatively static environment for strategy settings, being with slow technological and market innovations and stable industry structure. Today’s business environments, however, are so dynamic and shaky that no one can foresee them even half a year ahead. Harshly shook by rapid technological development and highly volatile market environments, today’s software development is under the constant necessity for swift and reliable development practices and market launch in appropriate timing. Given these dynamic settings, the time-scale that the position-based and the resource-based frameworks presuppose is too coarse to make swift and sound strategic decisions. In software development practices in rapidly changing environments, there is no time margin to determine strategic positioning against potential and/or existing competitors or to build unique resources that can serve as effective barriers for potential new entrants.

Here, there is another approach worth giving careful consideration in this context: Eisenhardt and her colleagues’ study on competitive strategy in high-velocity markets [5-8]. She has been focusing on business strategy and product development in rapidly changing environments, especially the computer industry and the internet business.

Her recent study [7] particularly looks at strategic management in the internet business with a case study of Yahoo!. Yahoo! is without doubt one of the most successful internet companies since the late 1990s and is continuously developing many novel services. Analyzing Yahoo!’s success from the position-based and resource-based frameworks of strategic management, she argues that both cannot offer firm reasoning:

Everyone recognizes the unprecedented success of Yahoo!, but it’s not easily explained using traditional thinking about competitive strategy. Yahoo!’s rise can’t be attributed to an attractive industry structure, for example. In fact, the Internet portal space is a strategist’s worst nightmare: it’s characterized by intense rivalries, instant imitators, and customers who refuse to pay a cent. Worse yet, there are few barriers to entry. Nor is it possible to attribute Yahoo!’s success to unique or valuable resources – its founders had little more than a computer and a great idea when they started the company. As for strategy, many analysts would say it’s not clear that Yahoo! even has one … If Yahoo! has a strategy, it would be very hard to pin down using traditional, textbook notions. (p. 108)

Based on this analysis, she argues that Yahoo! has a distinct strategy for competition in the high-velocity market; namely, Strategy as Simple Rules. The essence of this strategy is described in the sentence: “When business becomes complicated, strategy should be simple” (p. 116). Whereas the strategic logic of the position-based approach is “establish position” and that of the resource-based approach is “leverage resources,” the main
implication of strategy as simple rules is the importance of capturing “unanticipated, fleeting opportunities” (see Table 1).

Through several case studies on companies in fast-moving markets, she and her colleagues identify five simple rules that guide core strategic processes:

- **How-to rules**: Spelling out key features of how a process is executed
- **Boundary rules**: Focusing managers on which opportunities can be pursued and which are outside the pale
- **Priority rules**: helping managers rank the accepted opportunities
- **Timing rules**: synchronizing managers with the pace of emerging opportunities and other parts of the company
- **Exist rules**: helping managers decide when to pull out of yesterday’s opportunities

Yahoo!, for example, clearly holds four simple rules in developing and executing their products and services: 1) know the priority rank of each product in development, 2) ensure that every engineer can work on every project, 3) maintain the Yahoo! look in the user interface, and 4) launch products quietly. Sticking to these simple rules, Yahoo! succeeded in growing and surviving in the highly turbulent internet business market. The case of Yahoo! tells us that in high-velocity markets, firms must quickly find out unanticipated, fleeting opportunities and shrewdly seize them to grow faster than competitors. And yet they should not act in disorder. They must focus their strategy upon several important simple rules that swiftly guide significant business processes.

Each of the three frameworks of strategic management discussed above has its own strengths and weaknesses. For software development, which constantly interact with rapid flux of technological and market innovations, the traditional frameworks would not work. It is clear that the framework of strategy as simple rules practically fit today’s software development practices. The framework can also provide us with guiding principles for the ‘agile’ and ‘adaptive’ software development models, which well balance development practices between ‘excessive design’ and ‘no design’ [2, 10].

### 5. A Case of Internet Service Development

This section discusses a short case study that examines the applicability of strategy as simple rules in a particular software development context: internet service development. This empirical study is still on-going and thorough description and analysis of this case is far beyond the aim of this paper. Thus some initial results of the exploratory interview-based fieldwork are discussed here.

Internet services are now flourishing and distinctively different from package software businesses. It is a business model that provides internet users with some particular functions, capabilities, and/or useful services via the World Wide Web (WWW). They are normally used on web browsers like Internet Explorer and Firefox without installing particular software packages. Portals, such as Yahoo! and MSN, are a classic internet service that offers us useful navigation when connecting to the Web. Web search engines are another growing internet service. Google, a search engine giant, is now energetically developing and launching a wide range of internet services for free: Google Map, Google Earth, GMail, Google Base, and so on. For corporate users, the Application Service Provider (ASP) model is widely diffused and

<table>
<thead>
<tr>
<th>Strategic logic</th>
<th>Position</th>
<th>Resources</th>
<th>Simple rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategic steps</td>
<td>Identify an attractive market; Locate a defensible position; Fortify and defend</td>
<td>Establish a vision; Build resources; Leverage across markets</td>
<td>Jump into the confusion; Keep moving; Seize opportunities; Finish strong</td>
</tr>
<tr>
<td>Strategic question</td>
<td>Where should we be?</td>
<td>What should we be?</td>
<td>How should we proceed?</td>
</tr>
<tr>
<td>Source of advantage</td>
<td>Unique, valuable position with tightly integrated activity system</td>
<td>Unique, valuable, inimitable resources</td>
<td>Key processes and unique simple rules</td>
</tr>
<tr>
<td>Works best in</td>
<td>Slowly changing, well-structured markets</td>
<td>Moderately changing, well-structured markets</td>
<td>Rapidly changing, ambiguous markets</td>
</tr>
<tr>
<td>Duration of advantage</td>
<td>Sustained</td>
<td>Sustained</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>Risk</td>
<td>It will be too difficult to alter position as conditions change</td>
<td>Company will be too slow to build new resources as conditions change</td>
<td>Managers will be too tentative in executing on promising opportunities</td>
</tr>
<tr>
<td>Performance goal</td>
<td>Profitability</td>
<td>Long-term dominance</td>
<td>Growth</td>
</tr>
</tbody>
</table>

Table 1: Three approaches to strategy (Adopted from [7])
supporting a wide range of business operations. Salesforce.com, for example, delivers online Supply Chain Management (SCM) services to clients. The ASP model is now transforming its name into Software-as-a-Service (SaaS) with broader implications. Development of internet services could be interpreted as a distinct case for today’s software development since it is clearly in high-velocity markets. Whereas ordinary package software is deliberately planned, constructed, and launched in a relatively long time-span, internet services are developed in dynamic interaction with technological and market innovations. Internet services never become ‘final products’ and are rather constantly revised and added with new features and functions in dynamic response to user feedback. It is obvious that internet service development is one of the most lively and dynamic areas of software development.

A distinct case to be taken here is GREE, one of the largest and most successful Social Networking Services (SNSs) in Japan¹ (see Figure 2). SNS is an online community service whereby registered users can communicate and link with each other. Various SNSs are launched in the U.S. in 2003, including Orkut and Friendster. Having seen such a sprouting phenomenon, Mr. Yoshikazu Tanaka, a young Japanese engineer, started his own SNS, named GREE, in February 2004 as a completely private project. Starting with just 4 users at its launch, it gathered over 10 thousands registered users only within a month. As of March 2006, the user base expanded into 310 thousands. Currently, GREE is operated by a company, GREE Inc., that Mr. Tanaka founded in December 2004.

From a strategic point of view, the most distinctive characteristic of GREE’s service development is that when they upgraded the alpha version of the service in October 2005, they formally defined their service as the perpetual beta. This means that GREE will be an ever-evolving internet service that will never be finalized. Based on the above discussion on strategic management of software development, the service will continue a dynamic ‘dialogue’ with rapid technological and market innovations in order to constantly deliver better services to its user.

Referring to this policy, Mr. Tanaka said² that “we are actively taking advantage of various new technologies by continuously caring about user benefits and usability.” He as the founder also stated that in order to seize unexpectedly rising opportunities in the field, “swift decision making moment by moment is crucial.” He also said that he always paid great attention to “efficiency in decision making” and that “short time-span for launch of new services and/or functions” holds great impacts upon the whole operation. Moreover, in order to operate the service so swiftly, he insisted that “adaptive organizational structure against sudden environmental changes” is of paramount importance. All the notions he stated are strongly related to ‘speed’ and ‘flexibility’ and such distinctiveness of his management style clearly shapes GREE’s ‘simple’ software development strategy.

Here, one aspect of GREE’s development strategy should be taken in more detail; namely, the dual roles of beta versions. The original purpose of releasing beta versions is to evaluate and finalize the software’s usability and functions and to make final bug-fixing smooth through feedback from beta testers. For internet services that will never be finalized, however, a beta version is not only an actual product delivered to users but also media through which the next beta version is devised with dynamic negotiation with both technological and market innovations (see Figure 3). This duality of roles of beta versions is a totally unprecedented notion for software development but would be a critical factor affecting the quality of service, relational structure among stakeholders, and hence final profits of internet services.

As seen in the case of GREE, recent competition in internet services is unfolding in a totally different time-span from that of traditional software development. Such new concepts as Software-as-a-Service and perpetual beta will transform existing frameworks and approaches for software development. In order to cope with this rising reality, software development of today must deal with various new development practices from a strategic management perspective through dynamic interaction with technological and market innovations.

¹ http://gree.jp/ (Japanese only)

² The excerpts of the conversation with Mr. Tanaka were drawn from a 1-hour semi-structured interview with him conducted in September 16, 2005 at his office in Tokyo, Japan.
6. Concluding Remarks
The main objective of this paper was to explore strategic management issues in today’s software development. By applying several strategic management frameworks to software development practices, the paper sought a sound linkage between software engineering and strategic management. In summary, faced with increasingly dynamic and turbulent environments, software development must guide development practices through a constant ‘dialogue’ to technological and market innovations. Based on a brief review of the existing strategic management research, Eisenhardt’s framework of ‘Strategy as Simple Rules’ was discussed in terms of applicability to software development practices. And with a short case study of GREE, a Japanese SNS, the framework was proved to be well fit to software development especially internet service development in rapidly changing environments. Moreover, the analysis of GREE suggested that beta versions of the internet service played dual roles in its development processes, as a product and as media, and that the duality would be a critical factor for strategy making in internet service development.

There are some limitations in the discussions of this paper. Most of the discussions in the paper are still hypothetical and clearly need empirical validation with quantitative and/or qualitative methods. Furthermore, the paper only addresses internet service development as a case of today’s software development. Actual software development widely varies in its scale and settings. More detailed categorization of software development is clearly needed in future research.

7. REFERENCES