Method-based versus Software-based Design Innovation: A sequence-analytic simulation

Completed Research Paper

Zhewei Zhang
Temple University
zhang@temple.edu

Nicholas Berente
University of Georgia
berente@uga.edu

Youngjin Yoo
Temple University
youngjin.yoo@temple.edu

Abstract

Innovative design and development processes of all types are becoming increasingly important in today’s volatile environment. Firms that want to improve their design processes take either a method-based approach or a software-based approach. The former involves adopting a structured set of standardized practices, and the latter involves implementing a software-based design management solution as a guide to the process. To compare these two approaches, we use an empirically grounded simulation of the design processes of a large semiconductor manufacturer. We find that software-based approaches work significantly better in environments marked by high volatility, but no significant difference between the two in environments of low volatility.

Keywords
organizational routine, simulation, design innovation

Introduction

Design is a fundamental activity that successful organizations must perform (Simon, 1981; Baldwin, 2000). 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 that organizations produce become more complicated, digital, and global, organizations looking to innovate with their design processes often need to redesign existing design processes (Sanchez, 1996; Yoo, 2006). Furthermore, the pace by which organizations need to change their innovations 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, what we describe as “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 field, for example, where decades of standard practices are professionally dictated. The software development remains a notable exception – with a rich tradition of pluralistic methodological approaches (Hirschheim et al 1995). Other fields,

1 This paper is based on work supported, in part, by the National Science Foundation under grants VOSS-0943010 and VOSS-1120966. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
however, are undergoing 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 simulations 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 are often accompanied rather radical changes in the design routines as organizations often 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) underwent radical change in both the process and product of their design routines.

Though both approaches are important, in reality, many organizations emphasize one over another, but, as yet 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, we explore the relative efficacy of different approaches to design routines using a simulation methods rooted in empirical data from a major semiconductor manufacturer (DeltaChip). DeltaChip has been practicing both approaches across its global design teams for years. Through this simulation, we investigate how method-based and software-based approaches perform differently when firm faces different levels of environmental volatility. Further, we look to characterize differences between method-based and software-based process structures.

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

**Literature Review**

**Design Innovation: Methods vs. Software**

Methodologies and software form the basis of two critical approaches in design innovation. Methodology-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 to 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 in 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 (Ulrich & Eppinger 1995); innovation identification and management (Cooper 1990); integrated circuit design (Browy et al 1997); and architectural design (Gray & Hughes 2001); among others.

Software-based design innovation comprises the other broad approach to the improvement of design processes. Software-based approaches – although they may be more or less consistent with particular methodologies – 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 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 provided evidence that CASE tools can be associated with improved productivity (Norman and Nunamaker 1989; Banker and Kauffman 1991). Orlikowski (1993) emphasized the organizational change needed with the adoption of such CASE tools. Software-based design environments abound in other areas as well (although they are not nearly so thoroughly studies as CASE tools). These software-based design systems include product lifecycle management (PLM) systems for product design and development (Grieves 2006); innovation management systems (Dooley & O’Sullivan 2002); electronic design automation (EDA) software for integrated circuits (Khatri 2011); 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 (e.g., Tolvanen & Lytytinen 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 organization 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 we look to explicitly compare these approaches in two ways. First we look to understand the difference of their impact on performance given different levels of environmental volatility. Second we 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 organization, a routine can be considered as a series of closely related activities in order to accomplish certain objective. 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. Once a routine is executed, a feedback on the performance will be received by the organization. In response to the feedback, organization then will adjust how the routine will be executed next time, and the internal structures and dynamics will also be adjusted accordingly.

In Gaskin et al’s paper (2011), they proposed 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 event and allow deeper understanding on organizational routine.

**Methodology Oriented Design vs. Software Oriented Design**

**Case of DeltaChip**

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

In the more traditional SDD approach, a designer retains a higher degree of personal control over the design by employing softwares in each isolated design domain separately. The designers mainly rely on their own experience to make changes and improvements at methodology level and use softwares to generate the physical layout based on methodology design.

In a more recent PS approach, chip designers have started to adopt highly integrated automation techniques that can significantly improve the task performance as PS augments extensively human design judgment with computer-based algorithmic design capability. Unlike in SDD approach human experience plays the key role on design improvement, in PS the development of software drives the improvement of
chip design. In fact, there is a central department dedicated for tool design, and a key task of design teams using PS is to help the tool department to refine and improve the software. Our interview with PS approach based design team shows that the capability of tools used by them has been improved by 40 times in less than two years.

**Methodology Oriented Design**

The case of DeltaChip where two different design approaches have been used is not unique. We call the first approach, SDD in DeltaChip, a methodology oriented design approach. It emphasizes the methodology part of design and the improvement mostly comes from human's better and deeper understanding of underlying logic of design object. Human designer still dominates the design process as the central role. The software used in this approach usually is for implementing and realizing the methodology practiced by human. For example, an architect draws sketches in computer first then the software generates specified 3D models. Regarding to the design routine, methodology oriented design does not require frequent change and firm often is more conservative to the organizational change. Thus, the design routine normally is more stable over time.

**Software Oriented Design**

The second approach is what we called software oriented design, like PS in DeltaChip. As the capability of software and artificial intelligence has been significantly improved in the last decades, the role of softwares has also gradually shifted from assisting to (partly) replacing human judgement and implement. We see more and more firms adopted those heavily tool based design approaches in order to achieve better and faster design result. In this case, the rapid progress of software usually requires significantly change at design routine accordingly. For example, the typical irritation cycle for one SDD based team we interviewed at DeltaChip 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 methodology oriented approach, this approach is also typically associated with large amount of investment for higher computational capability and better algorithm.

**Environmental Effect**

In reality, firms may face different environment, which can greatly affect the performance of design process. Here, we use environmental volatility to identify two types of environment a firm may face: low volatile environment and high volatile environment. Snyder and Glueck (1982) suggest environmental volatility can be measured in two parts: market volatility and technological volatility. Since we are interested in performances of different design approaches, we mainly focus on technological volatility. Thus, in our case, environmental volatility mainly refers to the R&D competition in the industry. We propose two hypotheses on how environmental volatility will affect performances of two design approaches.

Hypothesis 1: In low volatile environment, methodology oriented design and software oriented design will yield similar performance improvement overtime.

In contrast, high environmental volatility gives more pressure from its competitors and is less tolerant of slow improvement or even failure. In this situation, designers are willing to try out different ways to solve problem without worrying too much on the failure. This allow designers to learn more from new design routines regardless of performance change and find better solutions.

Hypothesis 2: In high volatile environment, software oriented design will yield better performance improvement overtime than methodology oriented design.
Two-Stage Simulation Analysis

Model Setup

In this study, we set up a two-stage simulation model based on the data collected from DeltaChip. The first step involves simulation of design routine using Markov chain Monte Carlo (MCMC) method and the second step involves simulation of organization’s adjustment of the balance between exploration and exploitation. In our simulation, we 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 our empirical data, we believe 50 routines covers a long enough period to discover performance change pattern. Moreover, to overcome the randomness of MCMC method, we have performed 500 times of simulation runs and aggregated all results.

Stage One: Two-Level Markov Simulation of Design Routine

Following Gaskin et al’s organizational genetics approach, we propose a simplified model in our study. In this model, we decompose an event into two elements: activity as higher level element and affordance as lower level elements. We choose activity as the higher level element because it defines the objective and boundary of one event, representing the core of the event. Based on our empirical data, we define eight types of activities, shown in Table 1. We also include affordance which has 5 different values (Table 2) in our model. Affordance is a concept originally developed by ecological psychologist Gibson (1977, 1979) and has been increasing widely used in the information systems literature (Norman, 1999; DeSanctis, 2003; Markus, 2005) 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 same type, as same type activities may actually have different number and/or types of affordances. Also, cost and time of each event can be more precisely measured with additional information offered by affordance.

Table 1 Summary of Activity

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

Table 2 Summary of Affordances

<table>
<thead>
<tr>
<th>Pseudonym</th>
<th>Affordance Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Representation</td>
<td>Functionality to enable the user to define, describe or change a definition or description of an object, relationship or process</td>
</tr>
<tr>
<td>2</td>
<td>Analysis</td>
<td>Functionality that enables the user to explore, simulate, or evaluate alternate representations or models of objects, relationships or processes</td>
</tr>
<tr>
<td>3</td>
<td>Transformation</td>
<td>Functionality that executes a significant planning or design task, thereby replacing or substituting for a human designer/planner</td>
</tr>
<tr>
<td>4</td>
<td>Control</td>
<td>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</td>
</tr>
<tr>
<td>5</td>
<td>Cooperative</td>
<td>Functionality that enables the user to exchange information with another individual(s) for the purpose of influencing (affecting) the</td>
</tr>
</tbody>
</table>
We developed a two-level (activity-affordance) simulation model using Matlab based on MCMC method. The methodology of this model is shown in Figure 1. Transition between activities (step 3 and 6) is based on the Markov transition matrix of activities, which is generated from our empirical data. Transition between activities stops after a “Complete” activity is generated. Table 3 shows examples of generated routines of activities. “Transition” between affordances (step 2 and 5) is based on Markov affiliation matrix of affordance. Unlike activity, which is temporally sequential, affordances are not sequential in our model. Table 4 shows examples of generated activities of affordances. 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 (step 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, we have one activity transition matrix, six affordance affiliation matrix, six affordance frequency distributions and six affordance count distributions.

### Table 3 Activity Level Simulation Result: highlighted row shows a simulated routine with 25 activities

<table>
<thead>
<tr>
<th>Routine #</th>
<th>Activity Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3 2 4 7 6 4 2 6 5 6 4 6 5 6 4 6 4 5 8</td>
</tr>
<tr>
<td>2</td>
<td>1 2 2 2 4 6 3 2 5 7 6 4 6 4 4 4 4 4 4 5 6 4 4 6 4 4 4 8</td>
</tr>
<tr>
<td>3</td>
<td>1 4 4 5 6 5 7 2 4 6 6 4 6 3 8</td>
</tr>
</tbody>
</table>

### Table 4 Affordance Level Simulation Result: simulated affordances for highlighted routine in table 2.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 4 3 4 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2 5 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>5 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>1 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>5 4 1 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>5 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5 5</td>
<td></td>
</tr>
</tbody>
</table>
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 current routine yields a better performance than expected performance, then an organization is likely to learn more, vice versa. In the simulation model, activity transition matrix is updated through Equation 1. Parameter $\alpha$ represents organization’s design approach: smaller $\alpha$ suggests a more methodology oriented design approach is adopted, which also means a more conservative view that less change in design routine is preferred. $\Delta P$ represents the performance change over expected performance. Positive $\Delta P$ suggests a performance improvement. M and M’ are Markov matrix of the old routines and current routine, respectively representing organization’s existing knowledge and possible new knowledge. When holding $\Delta P$ constant, M will be more influenced by M’ as $\alpha$ increases, which means for the same performance change, organization with software oriented approach is more willing to adapt the routine change in the future. When holding $\alpha$ constant, new M will be more influenced by M’ as $\Delta P$ increases, which means for the same organization, it will learn more from new practice if the routine provide better performance. In the model, matrix is updated in the form of transition frequency (count of each type transition) matrix and then converted to transition possibility matrix. Table 5 and table 6 shows how transition matrix has changed from initial status after 99 updates.

Equation 1: $M = M + (\alpha + \Delta P)M'$; $\alpha \in [0,1], \Delta P \in (-\infty, +\infty)$
Evaluating the performance

Measuring the performance of a design practice can be complex. In traditional project management literature, cost, scope and schedule are three most important factors. A successful project is considered when an expected objective is met within time and cost constraints. As we define one event with activity and affordance, we first assign a weight to each affordance type, representing both the time and cost associated with that affordance, so that different activities of same type are actually differently represented in our model in terms of cost and time. We 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 consider as project A’ with same objective as project B but requiring 1/2 cost and time. In other words, our definition of performance here is more about efficiency of successfully completing a standardized design task. In our simulation model, a higher requirement of design routine then can be translated to requirement of smaller sum of affordance weights. We 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 our measurement, we then define a valid range of design performance. We believe only within certain range can the routine be considered as successfully finished. This helps us to eliminate the unrealistically high and low results due to randomness of MCMC.

Last, we compute the performance change as in Equation 2, ratio of current routine’s performance difference to the expected performance.

$$
\Delta P = \frac{(E(P) - P)}{E(P)}
$$

Environmental Effect

As we discussed before, environmental volatility may affect the performance over time differently for two approaches. To incorporate such environmental effect, we introduce an environment parameter $b$ into our 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 environment is highly volatile that competitors are rapidly improving their performance, firm will try to improving at the industry pace rather than own pace, then $b$ is set less than 1. In this environment, intense competition among firms requires 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 tolerated with slow and/or minor performance improvement. Together, we get the final updating Equation 3.

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

Results

The simulation tells us how organizations with different emphasis on innovations in design routines – specifically software-centered v.s. methodology-centered – will evolve overtime. Before we go into detail, we first run two-way factorial ANOVA test to check if approaches, environmental volatility and their interaction have significant effect on performance. As shown in Figure 2, we do find significant effects from such factors and their interactions.
In Figure 3a and 3b, $b$ is set to be 1.2, which represents a low volatile environment. Here, we don’t see significant performance differences between software oriented design and methodology oriented design. Both performances changed from 80s to 30s, and the change follows similar pattern as software oriented design performance in high volatile environment. Additionally, ANOVA test also show there is no significant effect from approach type. H1 is supported by our simulation result.

Figure 2 Two-way Factorial ANOVA Result for Performance Differences

<table>
<thead>
<tr>
<th>Source</th>
<th>Partial SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>20879.197</td>
<td>3</td>
<td>6959.73234</td>
<td>57.88</td>
<td>0.0000</td>
</tr>
<tr>
<td>Approach</td>
<td>1817.39382</td>
<td>1</td>
<td>1817.39382</td>
<td>15.11</td>
<td>0.0001</td>
</tr>
<tr>
<td>Volatile</td>
<td>14823.8027</td>
<td>1</td>
<td>14823.8027</td>
<td>123.28</td>
<td>0.0000</td>
</tr>
<tr>
<td>Approach#Volatile</td>
<td>4238.00053</td>
<td>1</td>
<td>4238.00053</td>
<td>35.24</td>
<td>0.0000</td>
</tr>
<tr>
<td>Residual</td>
<td>23568.3262</td>
<td>196</td>
<td>120.246562</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>44447.5232</td>
<td>199</td>
<td>223.354388</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3a Performance differences (lower is better) in low volatile environment
Figure 3b One-way ANOVA Result for Performance Differences in Low Volatility

<table>
<thead>
<tr>
<th>Source</th>
<th>Partial SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>252.428544</td>
<td>1</td>
<td>252.428544</td>
<td>1.53</td>
<td>0.2188</td>
</tr>
<tr>
<td>Approach</td>
<td>252.428544</td>
<td>1</td>
<td>252.428544</td>
<td>1.53</td>
<td>0.2188</td>
</tr>
<tr>
<td>Residual</td>
<td>16148.7069</td>
<td>98</td>
<td>164.782723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>16401.1354</td>
<td>99</td>
<td>165.668034</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4a and 4b show results when $b$ is set to be 0.8, which represents a high volatile environment; we clearly see the performance differences over time. Software oriented design approach outperformed methodology oriented design largely. Methodology oriented design did not provide significant performance improvement, the performance only changed from 80s range to 70s range, whereas software oriented design improved performance to 40-50 range. The one way ANOVA result also provide evidence that approach type do significantly affect firm’s performance over time. Thus H2 is also supported.

![Graph showing performance differences in high volatile environment]
In this study, we investigate if and how two approaches to design innovation, namely method-based and software-based, will perform different 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. To the contrary, when the environment is static, there were no significant differences between the two approaches to design innovation. Given that the introduction of new softwares often involve much higher costs including software, training, opportunity cost, as well as various implementation risks, one might argue that organizations will be better served by opting to a methodology-based innovation approach under a static environment. In other words, the benefit of software oriented design is not without cost. The practical implication from our study is that due to the high cost of IT investment, the adoption of software oriented design approach may not benefit firms when firm is not facing intense competition. The software oriented design should be best adopted by firms in a highly competing market.

The phylogenetic analysis in our study also provides additional insight to explain the performance differences. We show that the methodology-based approach produces much more divergent design routines in volatile environment than other all three situations. The reason for such high divergence is most likely due to the slow search process. In our simulation model, the design routine is improved via updating the transition matrix of different activities. Three factors determine the degree of updating in the way. The updating mechanism follows $M = M + (a + ((b\bar{P} - P) / b\bar{P}))M'$. 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 firm with method-based approach has fewer chance of changing design routines, it is less likely to benefit from performance improvement due to routines changes. This is especially serious in volatile environment since there is lowered chance of getting significant performance improvement. While in static environment, the chance of getting significant performance improvement is much higher so that the disadvantage of method-based approach is less severe. This finding provides potentials for management at firms using method-based approach to improve their performance in volatile environment. The key is to encourage designer open to new opportunities, to increase $a$. The large $a$ in software-based approach is brought by the radical improvement of software capability forcing the organizational routine to change accordingly. However, there are also many other ways to increase $a$: an open minded atmosphere, a corporate culture more tolerant with failure and etc. In science, many important findings and successes are actually caused by failure. The invention of Post-It in 3M is a classic example. This is also have been observed in real world, in recent year, as chip-design moves to more mobile-computing oriented market, the competition has

### Figure 4b One-way ANOVA Result for Performance Differences in High Volatility

<table>
<thead>
<tr>
<th>Source</th>
<th>Partial SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>Prob &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>5802.9658</td>
<td>1</td>
<td>5802.9658</td>
<td>76.65</td>
<td>0.0000</td>
</tr>
<tr>
<td>Approach</td>
<td>5802.9658</td>
<td>1</td>
<td>5802.9658</td>
<td>76.65</td>
<td>0.0000</td>
</tr>
<tr>
<td>Residual</td>
<td>7419.61932</td>
<td>98</td>
<td>75.7104012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>13222.5851</td>
<td>99</td>
<td>133.561466</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Discussion**

In this study, we investigate if and how two approaches to design innovation, namely method-based and software-based, will perform different 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. To the contrary, when the environment is static, there were no significant differences between the two approaches to design innovation. Given that the introduction of new softwares often involve much higher costs including software, training, opportunity cost, as well as various implementation risks, one might argue that organizations will be better served by opting to a methodology-based innovation approach under a static environment. In other words, the benefit of software oriented design is not without cost. The practical implication from our study is that due to the high cost of IT investment, the adoption of software oriented design approach may not benefit firms when firm is not facing intense competition. The software oriented design should be best adopted by firms in a highly competing market.

The phylogenetic analysis in our study also provides additional insight to explain the performance differences. We show that the methodology-based approach produces much more divergent design routines in volatile environment than other all three situations. The reason for such high divergence is most likely due to the slow search process. In our simulation model, the design routine is improved via updating the transition matrix of different activities. Three factors determine the degree of updating in the way. The updating mechanism follows $M = M + (a + ((b\bar{P} - P) / b\bar{P}))M'$. 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 firm with method-based approach has fewer chance of changing design routines, it is less likely to benefit from performance improvement due to routines changes. This is especially serious in volatile environment since there is lowered chance of getting significant performance improvement. While in static environment, the chance of getting significant performance improvement is much higher so that the disadvantage of method-based approach is less severe. This finding provides potentials for management at firms using method-based approach to improve their performance in volatile environment. The key is to encourage designer open to new opportunities, to increase $a$. The large $a$ in software-based approach is brought by the radical improvement of software capability forcing the organizational routine to change accordingly. However, there are also many other ways to increase $a$: an open minded atmosphere, a corporate culture more tolerant with failure and etc. In science, many important findings and successes are actually caused by failure. The invention of Post-It in 3M is a classic example. This is also have been observed in real world, in recent year, as chip-design moves to more mobile-computing oriented market, the competition has
become much fiercer. The time to market of a product become much more critical for a product success. In the latest project, the design cycle has been shortened from 36 months to 18 months. As a result of such change, the SDD approach, methodology-based approach, has been completed phased out and replaced by PS, a software-based approach. The advantage of improving design efficiency brought by software-based approach becomes superior in such turbulent environment.

From the organizational learning perspective, method-based approach and software-based approach in fact represent two major mechanism-exploitation and exploration. On one hand, various design routines of method-based approach mostly derive from a few major routines, suggesting this approach attempt to exploit full potentials of few possibilities. On the other hand, tool-based approach tent to learn from exploring enormous possibilities with fewer constraints as no dominant routine can be identified in the phylogenetic trees.

**Conclusion**

In this study, we try to understand the different performances of two major design innovation approaches: methodology-based approach verse software-based approach. We ground our research with our empirical works with industry partners. Based on the empirical data, we build a two-stage simulation model to forecast the evolution of design routines in different situation. Our findings provide new insight on the effect of design approach in different environment and the reason behind such different effects. We also offer practitioner recommendations on decision on approach choice when facing different competition environment and suggestions on other possible ways to improve performance without altering design approaches.

**References**


