Technological designs for retention and reuse

Raghu Garud
Associate Professor of Management, Leonard N. Stern School of Business, New York University, 713 Tisch Hall, 40 West 4th Street, New York, NY 10012, USA

Arun Kumaraswamy
Leonard N. Stern School of Business, New York University, 7-150 Management Education Centre, 44 West 4th Street, New York, NY 10012, USA

Abstract: At one time, firms designed technological systems for obsolescence. Increasingly, however, firms are designing technological systems to facilitate the retention and reuse of components. This paper offers guidelines for how technological systems can be designed for retention and reuse, and explains why retention and reuse of components have become important.

Keywords: Technological systems; reuse; modularity; upgradability; learning.


Biographical notes: Raghu Garud received his PhD in Strategic Management and Organization from the University of Minnesota. He is currently an associate professor of management at the Leonard N. Stern School of Business, New York University. His teaching and research interests lie in exploring the intersection among technology, organizations and strategy. Currently, he is co-editing a book titled ‘Technological Entrepreneurship: Oversights and Foresights’ to be published by Cambridge University Press.

Arun Kumaraswamy is a doctoral candidate in Strategic Management at the Leonard N. Stern School of Business, New York University, New York. His research interests include investment strategy and competitive dynamics in high technology industries. He has published several papers on these and related topics. Currently, Arun is working on his dissertation which uses a real options perspective to understand high-technology firms' R&D investments and management.

1 Introduction

Chronicling the advent of a 'throw-away' society, Alvin Toffler [1] noted that technological systems were increasingly being designed for transience. For instance, there was a time when automobiles were replaced every four or five years because they had
outlived their useful life expectancy. Few, if any, components were salvaged for reuse, and most such automobiles were scrapped.

These 'economics of transience' arose from the difficulties and costs associated with the recovery and reuse of components; it was simply easier and cheaper to replace the entire system to obtain higher performance. This statement was as much true for customers as it was for vendors who designed and marketed technological systems.

Recently, however, technological change has become too rapid for customers and designers to abandon entire designs and systems. It is now more economical to design technological systems in such a manner that components of an existing system are retained even as other components are replaced to yield performance gains. How might firms design technological systems for component retention and reuse?

To address this question, we first explore the construction of technological systems and introduce three system attributes that are important in their design — integrity, modularity and upgradability. We suggest that retention and reuse are facilitated to the extent that technological systems are modularly upgradable. To the extent that a system is modularly upgradable, it is easier to integrate new components into the system without having to design the entire system afresh. We illustrate the implementation of modularly upgradable designs using Object Oriented Programming (OOP). In conclusion, we discuss the implications of designing reusable system components for the preservation of knowledge and learning.

2 Attributes of technological systems

Technological systems consist of many components that together provide utility to users. System performance is dependent not only on the performance of individual components, but also on the extent to which these components are compatible with one another [2,3,4]. Although individual system components are designed to provide a high level of performance, a lack of compatibility among them will result in sub-optimal system performance. In other words, incompatibility between system components can compromise system integrity.

During early stages of industrial activity, designers ensured high system integrity by 'custom-designing' individual components for compatibility. Lacking standard gauges, master craftsmen forged and cut metals to create system components using general purpose machine tools. These components were then 'fit' together through repeated rework at the joints or interfaces [5,6]. Each system 'crafted' using this approach was unique, thereby limiting the learning retained and transferred over time. Learning focused more on developing general expertise in metal working than on manufacturing specific components or systems. This learning was transferred from one generation of craftsmen to another through apprenticeship — a learning-by-association process that limited the size of job shops engaged in craft production.

Advances in manufacturing processes, and standardization of component specifications made it possible to manufacture high-integrity systems without necessitating extensive rework. Advances in manufacturing processes such as the use of standard gauges enabled mass production of components that could then be assembled easily without any rework. These advances resulted in specialization, or modularity-in-production [7] that reduced manufacturing costs through learning by doing [8].
Standardization of system components involves codification of specifications pertaining to the form and function of components and the interfaces among components. This standardization results in modularity-in-design [7]. Modularity-in-design facilitates a process of learning-by-studying; vendors can create system configurations with a wide range of functionality and performance by experimenting with various combinations of components [9,10].

In some cases, modularity-in-design may also lead to modularity-in-use [7]. Modularity-in-use allows customers to customize system configurations to suit their specific requirements. To the extent that modularity-in-design also leads to modularity-in-use for customers, it becomes possible for both vendors and customers to benefit from a different type of learning – learning by using [11].

Learning by using involves the incorporation of feedback on how a system performs when customers use it [12]. Information on what problems exist and which system features are most useful, if fed back to the vendor, can create important knowledge. As long as modularity exists in production and in design, the vendor can employ this knowledge to make appropriate changes in system design and selected production processes easily, thereby satisfying changing customer preferences and requirements [13].

<table>
<thead>
<tr>
<th>Type of modularity</th>
<th>Associated learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modularity-in-design</td>
<td>Learning by studying</td>
</tr>
<tr>
<td>Modularity-in-production</td>
<td>Learning by doing</td>
</tr>
<tr>
<td>Modularity-in-use</td>
<td>Learning by using</td>
</tr>
</tbody>
</table>

The degree to which learning occurs depends on whether component and interface standards are specific to a single firm or common to all firms within an industry. In the former case, modularity is restricted to the firm’s own product lines, thereby limiting the benefits of learning to only that firm. On the other hand, if interface standards are defined at the industry level, learning and benefits flow to all firms within the industry.

Industry-level component and interface standards allow firms to manufacture compatible components. Such multi-vendor compatibility allows customers to mix and match components from different vendors to create a variety of system configurations that match their specific performance requirements [14]. This enables 'learning by using' for all firms within the industry. At the same time, the benefits of 'learning by doing' are available for those system components that are generic to different system configurations and, therefore, can be mass-produced. In this manner, modular designs combine 'learning by doing' and 'learning by using', thereby enabling mass customization [15,16].

Excessive modularity, however, can compromise system integrity. For instance, with each new effort to modularize the system, the number of interfaces increases. As the number of interfaces increases, the possibility of performance losses at the interfaces also increases. Furthermore, excessive modularization increases the complexity of the design. It is for these reasons that designers use techniques that clump a set of components into a natural agglomeration, thereby reducing the number of interface specifications that have to be dealt with at any given time.
Designing high-integrity and modular technological systems, however, is not sufficient to maximize learning in dynamic environments. In such environments, an additional system attribute, *upgradability*, becomes important. Without upgradability, new product offerings will destroy earlier designs through a cannibalization process that is difficult to sustain indefinitely, for both vendors and customers.

Upgradability is possible only if a system possesses sufficient technological degrees of freedom. To understand how technological degrees of freedom can be created, it is important to appreciate the hierarchical organization of components within a technological system [17,18,19]. Component choices at any level of the system hierarchy outline operational boundaries for lower-order components and sub-systems. It is possible to choose higher-order components that possess performance capabilities that are not fully exploited at early design stages. These unused technological degrees of freedom provide designers the latitude to increase system performance through innovations in lower-order components. Because innovations occur only in lower-order components, designers and customers are able to upgrade easily to the next generation of systems.

Such exploitation of unused degrees of freedom represents a 'compulsive' process wherein improvements in system performance are accomplished through advances in specific components even as the full potential of other components remain to be exhausted [11]. However, the benefits of reusing most components and replacing only a few must outweigh the costs associated with such reuse and selective replacement. In the next section, we explore what these benefits and costs are, and how we might design systems for retention and reuse.

### 3 System designs for retention and reuse

Benefits from retention and reuse can be either direct or indirect. Direct benefits include the reuse of components and knowledge concealed in them. Indirect benefits include cost savings during qualification, manufacturing, and the development of complementary technologies.

These benefits have to be balanced against *performance slippage, incorporation costs* and *initial design costs* that arise from designing for reuse [7]. Performance slippage occurs because newly incorporated components may be incompatible with reused components, thereby compromising system integrity. Incorporation costs arise when new components have to be modified to make them compatible with reused components. Initial design costs are up-front costs that arise because designers have to define interface standards and build sufficient degrees of freedom into components to enable their reuse.

Performance slippage can be minimized by using *gateway technologies* or by creating *transient designs*. Gateway technologies enable the co-existence of incompatible components within a system through the use of adapters and converters [20]. Because they imply the development and use of additional components, gateway technologies give rise to higher design and incorporation costs. Furthermore, they seldom restore system integrity completely. Therefore, gateway technologies may not be the best way to increase the retention and reuse of components. However, for many systems that were originally designed for obsolescence and incompatibility, gateway technologies remain the only way to reuse components [21].
A better approach for enhancing retention and reuse is to create what we label as transient designs [22]. Transient designs involve the creation of systems that can keep pace with technology evolution. In addition to being upgradable, transient designs are also modular. Such modularity reduces the cost of incorporating new components into the system even while retaining the basic technological platform [23,24]. To the extent that a system is modular, incorporation costs are limited to removing incompatibilities that have arisen since the implementation of the currently prevalent interface standards.

Depending on the ease of incorporation, three types of transient designs can be obtained – add designs, replace designs and transmutation designs [25]. Add designs improve system performance by enabling the addition of new components. To the extent that new components are modular and the existing system has unused technological degrees of freedom to allow for the addition of new components, add designs result in the retention of nearly all existing components. In contrast, replacement designs entail the substitution of one component for another, and therefore the level of retention is lower than in add designs. However, the old component may be refurbished and used again, thereby recovering some of the value lost due to replacement.

The third type of transient design is transmutation design. Transmutation designs accommodate changes in overall system architecture without compromising system integrity. Such designs require coordination at the system-architecture level so that a seamless synthesis of old and new technologies can be accomplished to create new functionality in the future. Therefore, transmutation designs are the hardest to create, and require higher up-front expenditure in the form of initial design costs. But, once created, transmutation designs enable high retention of learning and knowledge through the reuse of components across generations and across product lines.

3.1 Retention and reuse in Object Oriented Programming

The above design principles which enable retention and reuse of components are well illustrated by Object Oriented Programming (OOP). OOP is a method of creating large software programmes from a library of small prefabricated modules, thereby preserving knowledge and enhancing software productivity.

OOP can be described best by three design principles — encapsulation, inheritance, and polymorphism. Encapsulation refers to the creation of reusable blocks of code called objects. Each object is a module that contains all relevant programs and data to represent a real world entity [26]. The notion of encapsulation is similar to modularization in transient designs.

These objects are arranged in a hierarchy based on their level of abstraction. The level of abstraction is akin to the notion of technological degrees of freedom in transient designs. Specifically, the greater is the level of abstraction, the greater is the hierarchy's technological degrees of freedom [27]. This hierarchical arrangement by level of abstraction promotes inheritance or the transfer of traits from objects defined at a given level of the hierarchy to objects at all lower levels. The third design principle, polymorphism, is the ability to address different objects simultaneously using the same message [28]. Such a polymorphic design principle is akin to the notion of clumping in transient designs.

Encapsulation, inheritance and polymorphism lead to the reuse of objects in OOP and increased programmer productivity. In general, studies have shown that object-oriented
programming improves programmer productivity by a magnitude of two to five times [29]. Because objects are modularized and can be reused, programmers do not have to redesign new software programmes from scratch. For instance, Shearson Lehman Brothers Inc. built OOP software to manage its operations and cut development costs by 30% through the reuse of objects [30].

Additionally, inheritance and the polymorphic design of software programs allow new objects to be created using fewer lines of code. As a result of hierarchical organization of objects and inheritance, any change at a given level of the hierarchy will be reflected in all lower levels. Therefore, only the differences in characteristics between successive levels of the hierarchy need to be coded. Similarly, polymorphism reduces the size of the program by allowing an object to address several other objects simultaneously and trigger required responses from them. For instance, Brooklyn Union Gas Company created an object-based program that was 40% smaller in size and, at the same time, performed more functions in terms of customer service than its existing mainframe software [30].

Moreover, inheritance allows a programmer to "mutate" an existing object into a new object by redefining characteristics at higher levels of the hierarchy. Similarly, polymorphism allows the programmer to increase the capability of any object to address many objects simultaneously by just making localized changes within that object. Thus, inheritance and polymorphism also ensure easy upgradability and faster development time. For example, the US Marine Corps reduced prototype time from the normal six-to-eight weeks to just two weeks by applying OOP techniques in developing Ada-based IS applications [31].

4 Discussion and conclusion

We began this paper by asking how technological systems may be designed for retention and reuse. In answer, we offered modular upgradability as a guiding design principle. We illustrated how modular upgradability enhances retention and reuse of software modules in the case of object-oriented programming.

Modularly upgradeable transient designs such as these are finding increasing use not only in OOP but also in the design of many complex systems. For instance, Sun Microsystems' Sparcstation10 family of workstations [32] and Texas Instruments' PRISM methodology [33] for designing and manufacturing semi-customized integrated circuits represent transient designs employed in the computer hardware and electronics industries.

In the automobile industry too, car manufacturers have been resorting to platform designs to reduce development time and the cost of offering a variety of models [34]. For instance, in recent Honda models, components designed for one model can be reused with minor changes in several other models [35]. Similarly, by reusing up to 70% of components designed for older models in new designs and simplifying the design of complex subsystems such as bumper assemblies and cooling systems, Toyota saved over $500 million in designing a new four-wheel drive vehicle called RAV4 [36].

In all the above cases, modularity provides system designers with the flexibility to substitute only certain system components even while retaining the others. At the same time, upgradability provides designers with the opportunity to work on an already established technological platform thereby preserving their core knowledge-base [23]. In
this manner, modular upgradability leads to the preservation of knowledge across
generations and simplifies the task of coping with very short product life cycles.

Besides enabling the preservation of knowledge across generations, modular
upgradability creates new knowledge that enhances, rather than destroys, existing
knowledge. This competency enhancing knowledge [37] arises from experience as
designers gain a deep appreciation of the future potential of the base technological
platform. In this case, learning consists of appreciating:

(1) which aspects of the platform will lead to future improvements,

(2) which aspects will lead to dead ends, and

(3) how new lower order components fit in with the base platform.

Modular upgradability leads to reuse in another way. Firms can conduct customer trials
and incorporate suggestions for improvement by replacing only certain system
components while retaining others. Rosenberg [11] points out that this process is essential
for the evolution of complex multi-component systems because system optimization and
problem detection can occur only through large scale customer trials. Insofar as the
system is modularly upgradable, designers will find it easy and economical to carry out
these modifications - a process that Wheelwright and Clark [23] label as rapid
prototyping.

Finally, modular upgradability is becoming important from an environmental
perspective as well. Of late, several firms including office equipment manufacturers,
computer vendors and automobile manufacturers are adopting a new technique called
design for disassembly [38]. For instance, BMW had re-conceptualized the design of its
automobiles so that body panels may be collapsed and removed easily. Such a design
enables easy disassembly and the recovery of components for refurbishing, reuse or easy
recycling. Additionally, the ‘Big Three’ US manufacturers are evaluating the ease of
disassembly of their automobiles as part of a joint study and plan to introduce design
changes to increase component recycling and reuse.

To sum up, retention and reuse are important design attributes in contemporary
environments. Retention and reuse can be accomplished by designing modularly
upgradable technological systems. However, such a design focus raises several
organizational issues. For instance, the organizational structure needs to be modified
considerably to promote learning and knowledge sharing. Additionally, the control and
incentive systems need to be changed so that designing for reuse and actual reuse are
monitored closely and rewarded. These issues merit a detailed discussion which we
provide in a different paper [21].

Acknowledgment

Support for this research has been provided in part by a grant from Deloitte and Touche
Management Consulting Group. We thank Donald Curtis of Deloitte and Touche for
sponsoring this research.
References and notes

12. Learning by using is important specially for complex systems in whose case the vendor cannot test system functionality under every possible mode of operation. Moreover, customers usually find innovative uses for the system that go beyond the originally intended use.
13. This practice is evident in the computer hardware market. Typically, Sun Microsystems rushes to the market with products that are only half done. Through customer trials, Sun identifies bugs and shortcomings, and offers newer versions that rectify these. Sun's systems are modular, thereby allowing customers to incorporate these changes easily into their existing systems. This practice is also evident in the computer software market in which complex software programs such as operating systems go through a series of tests (beta tests) at customer premises. A recent example is the beta testing of Microsoft's Windows 95 operating system, with each round of beta tests allowing Microsoft to uncover and rectify bugs and incompatibilities.


24 Transient designs enable retention and reuse by creating options for the future. Investment in the creation of modularity and upgradability during the initial design stage represent the price of the option. Incorporation costs incurred to prevent performance slippage during the addition or replacement of components represent the exercise price of this option. Insofar as the price of the option plus its exercise price are lower than the benefits expected from these flexible designs, economies may be realized.


26 For instance, in a children's software programme dealing with pets, each animal - a dog, a cat, or a frog - will be defined by an object. The dog object will contain all programmes and data required to describe a dog.

27 In the children's software programme mentioned earlier, the definition of a 'dog' object at a higher level in the hierarchy allows the programmer to define objects at lower levels that stand for different species of dogs like the 'poodle' or 'Alsatian'.

28 In our children's programme, a 'talk' message from an object will obtain a 'meow' response from a cat, and a 'woof' response from a dog.


31 Information obtained during telephonic conversations with an industry analyst.


33 Texas Instruments' (TI) PRISM methodology involves the reuse of modules from TI's circuit library to create standard and semi-custom products. Reuse of modular circuits leads to lower qualification costs and reduced product development cycle time and costs.


