Everything you always wanted to know but were afraid to ask about the decision-making process.

BY JAN DAMSGAARD AND JAN KARLSBJERG

Seven Principles for Selecting Software Packages

AROUND THE MID-1950S, in the early years of commercial use of computers, all software systems were developed in-house. There was no software industry in existence at that time.¹⁵ As the software industry formed over the next few decades, many organizations outsourced their software development to specialized software

suppliers. Most software products were, however, still developed as unique systems for each organization; that is, there was little standardization. The next step occurred when that software producers developed their own proprietary software in order to capture economies of scale in developing the software once and then selling it to multiple customers.² This standardization process also benefitted software buyers by lowering transaction costs and risks, as it was now possible to choose among a proven set of applications. Moreover, standardization gave both producers and buyers of soft-

ware a way to capture and black-box best practices by embedding it into the standardized components of the systems.¹⁶ Next in the standardization process was a step away from proprietary standard systems that essentially locked customers to a single software producer to open software standards.³⁷ In principle, software built on open standards allowed customers to source from any supplier that could supply software in accordance with open standards (for example, Java- and XML-based systems).

Open standards meant that prices dropped and functionality was en-

hanced, which resulted in a mass market for many software application types. In addition, software producers had enough resources to make their software even more general-purpose oriented with larger feature sets that were organized into a product.⁸ Software became even more standardized, and in the process, many local markets were annexed into global markets. For example, word processing software was no longer produced specifically for a particular profession or industry or nation;²⁶ instead, an almost universal office suite emerged, such as, Microsoft Office. The generalized software products could be configured in various ways (for example, program parameters, macro functionality, language support, and so on) to suit special needs among customers. These highly configurable general purpose software products came to be known as software packages.²³

Until recently in the IS academic community, there has been a tendency to focus on traditional studies of software development and implementation of large custom-made systems.^{20,24} This has been despite the leading trend that organizations use “shrink wrapped” systems³¹ where the core functionalities of the software are identical across all implementations in dozens, thousands or even millions of different organizations.¹⁵ When it comes to managing the process of identifying and evaluating packages, the IS academic community has been almost silent.

The aim of this article is to provide practitioners with a grounded set of principles to guide the selection of software packages. By principles, we mean a set of fundamental ideas that summarize important insights in software package acquisition that are not (yet) embedded into the practice of buying software. The principles are interdependent and together they form a whole that is larger than the sum of the parts. Similar to Klein and Myers’ argument,¹⁹ the use of all principles is not mandatory, but in each case it must be judged whether, how, and which principles apply to a specific situation.

Packaged Software

Packaged software is a category of information systems for which all

implementations are essentially identical; that is, the main functionalities are common to all adopters. While the core components of a package are identical across all user organizations, the implementation into an individual organizational information infrastructure is usually configured in some manner to fit the requirements of the organization.^{1,17,22} For the purpose of this article, we define a standard software package as: a collection of software components which when combined perform a set of generalized tasks that are applicable to a range of users. As a package is adopted by many, it forms a standard because the core components are identical across all of its installations. The software package may be configured or customized to make it fit with specific requirements unique to the concrete implementation. This is accomplished by setting program parameters, installing add-on modules, or building interfaces with other software systems. Within an organization, the growing importance of system interconnec-

tions means the choice of software package has wide ripple effects for other parts of the organization whose software packages, implementations, and interests may not originally have been identified or considered in the decision process regarding a new software acquisition.

Packages are often referred to as “commercial off-the-shelf” software,³¹ but open source systems (for example, Open Office) or other types of nominally free software, for example, Firefox²¹ or Internet residing systems (for example, Google Apps) are other examples of packaged software. Some standard software packages require little adjustment on the part of the user before they can perform (for example, Internet Explorer), while other software packages are mere tools or platforms on top of which specific functionalities required by the user can be implemented (for example, ERP systems).⁸ Some setups of parameters may be common among several customers, in which case the producer can offer *standard solutions* on top of which only site-specific con-

figurations need to be made.³⁵ For example, the ERP producer SAP provides more than 25 industry solution portfolios for large enterprises that embed best practice (for example, SAP for oil and gas).

Seven Guiding Principles for Selecting Software Packages

Here, we present the guiding principles for making a better informed choice when selecting software packages. The first principle we label the founding principle because it is fundamental to the other six. For each principle we provide examples that illustrate its importance.

The seven principles were derived empirically from a field study and from our understanding of software acquisition. The field study approach provided us with in-depth knowledge of a number of standards decisions made by actual organizations. The focal company had more than 50,000 employees, and we followed its software acquisition processes and standard choices over three years. The

field study was conducted using semi-structured interviews. The persons interviewed were five senior directors with knowledge of—and some power to influence—software acquisition. To broaden our knowledge base, we also carried out 34 interviews in 13 other organizations. The interviewees were CIOs, CFOs, and general managers, and were deliberately chosen because of their high experience with software package selection processes. All interviews were recorded and transcribed and thematically coded. The longitudinal approach meant that a theme identified in one interview could be further investigated and validated in subsequent interviews.

A second source of inspiration was information about particular software standards and packages, vendors, historic data about system compatibility, market shares, and mergers and acquisitions. Yet another source of inspiration came from participating in an industry network in the late 1990s where representatives from 60 companies met bimonthly to share

their knowledge and experience with corporate intranets. During the three years the nature of intranets changed from being home grown, to a situation where a few local software companies vigorously fought over market share, to a situation where intranets were built upon international standards and readily available from multiple software houses. The seven principles have now been presented and critically reviewed at numerous IT managers' conferences, and we are indebted to the participants for many of the examples that illustrate the principles.

The Fundamental First Principle: When you buy packaged software you join its network.

Prior to the emergence of packaged software, any organization that was using software in effect committed itself not only to a software product but also to a particular software producer's continued ability to deliver new functionalities, as organizational requirements evolved and new technology became available. In the present day, most of

these commitments and dependencies have evolved from local software producers to global standard software packages that can be sourced from, and configured by, many independent software vendors with the necessary competences and technical skills.

The users and producers of a software package constitute a network of parties that share a common interest in its destiny.³⁴ The network is virtual, in the sense that the members probably do not know each other but nevertheless share a common interest in protecting their investments and ensuring the continued evolution of the package. The network indirectly also has other interests in common; for example, the training and education of personnel.³⁴ An organization's purchase and implementation of a particular software package thus means that the organization has joined the network associated with the software package, and the level of commitment is equal to the size of the investment (buying and configuring the software and the training of personnel, and so on). To a large extent, the investment represents sunk costs,¹⁰ which make risk mitigation activities even more central.

The network around the package has implications for the purchasing decision and has to be considered as part of the investment decision. Beyond the immediate network of users and producers, the extended network includes vendors, standard setting institutions, government authorities and other compatible software products. It is imperative to choose to participate in the network that is perceived to provide the best long-term benefits as the organization, the network and package co-evolve. In the network the distribution of power and influence depends chiefly on who controls the package and thereby its evolution. In the case of most software packages, the producer wields the greatest power over the proprietary software network, as they own the rights to the package outright and thus control its further development. The producer's power can be challenged if users unite to influence the producer or even challenge the producer's ownership; for example, by reverse-engineering the package's functionality. As an example of influence, the pressure from power-

The users and producers of a software package constitute a network of parties that share a common interest in its destiny.

ful users has repeatedly postponed the sunset date of Windows XP.

Open source packages, on the contrary, are not owned by a single entity; instead, the software is designed specifically to promote shared ownership.^{25,29} Open source software can appear unattractive and risky to some because there is no central point of control from which advice about the software package and its future development can be sought. Others view these properties as strengths since they protect the standard package from the opportunistic actions of profit maximizing software producers. We shall not conclude the heated debate over open source here, but merely emphasise that organizations adopting a software package need to be alerted to the intimate connection between a software package and its associated network.

Principle Two:

Take a long-term perspective:

Look ahead but reason back.

Many choices made in the early stages of an organization's use of computers have turned out to have surprisingly long-lasting consequences, as both software and data standards have been shown to be very persistent.²⁰ Many application types have historically developed in an evolutionary manner, where the first simple implementations were custom built by innovators, and then spread to a small number of early adopters. As the application type benefited its adopters, competing systems became available on the market, and finally the application type became a commodity, possibly to be bundled with other software application types into larger software packages. A similar evolution trajectory will likely describe the development of future application types that first appear as isolated systems. As a consequence, organizations must take a long-term perspective and envision a more complex and connected future, or else they risk implementing tomorrow's legacy systems.

We emphasize this long-term perspective of software packages. As the pace of change in the computer industry reduces the effective lifespan of most hardware and software to a few years, the organizational data and the

standards that define them are more durable.⁵ An organization's standard package choice therefore involves participation in networks that may last a decade or often longer. Shapiro and Varian³⁴ argue that when buying standard technology we should look ahead but reason back, noticing the network and the evolution process that produced it. We applaud and echo this advice that is valid also when selecting packaged software. This principle is useful to include when comparing a proprietary software package from a local vendor with that of a software package built upon an open global standard.

Principle Three:

When choosing packaged software, there is safety in numbers.

One route to mitigating the perceived risk in purchasing packaged software is to choose a package based on its historical and current success, as measured by the financial success of the software package's producer and the size of the associated network. *Flocking behavior* is a low risk strategy that is worth pursuing for software support of non-core functionality and for companies that consider themselves followers. Below, we describe two scenarios representing opposite outcomes of a competition between software packages; namely, *blind alleys* and *one-way streets*.¹²

The *blind alley* scenario refers to the situation where an organization has adopted a package that is losing its market share to competing packages. David¹² uses the term "angry orphan" to describe the situation of the losing package. He points out that such products often show a sudden rapid development when they are losing the battle. For example, the greatest speed of innovation in sail ships happened as the steam engine challenged the sail as the leading propelling technology on sea voyages. Despite the sudden and remarkable development, sail boats never really challenged the inevitable change to steam engine boats. In a similar manner, the losing software package might undergo rapid developments, but shrinking network effects make the downward spiral inevitable. In a special case of the blind alley scenario, the losing package manages to capture a niche market network where it may sustain itself for years—or even

perpetually, giving organizations the choice of staying with the incumbent producer or giving them time to look for migration paths toward a standard package with more perceived vitality.¹⁷

The *one-way street* scenario describes the situation where the organization is left with little choice when it comes to buying upgrades or expansions to the package. This is the case when the purchase of a particular package in effect obliges the organization to place future purchases with the same software family because the product has low compatibility with other families of software or packages. In this situation, the organization may find itself chained to the producer because the costs involved in switching to another package are prohibitively high, and the organization is in effect locked-in.¹¹ This is quite common for ERP systems where once the initial choice between (for example, Oracle Financial Systems and SAP), has been made, it becomes prohibitively expensive to switch. Sometimes, a package may be so successful in the market that there are few—if any—viable alternative products available to the organization, an example of which is the current choice of operating systems for PCs being limited to Microsoft Windows, creating a near monopoly. However monopolies are constantly challenged and they are often short lived in the software business, as reported by Chapman⁷ who narrates the story of how WordPerfect lost its near monopoly and how other software packages such as Netscape and dBase lost their lucrative position in the market.

Principle Four:

Focus on compatibility and beware of false gold.^a

Because of the long life expectancy of organizational data stored in some (often proprietary) format (see Principle Two), backward compatibility between software systems becomes a major factor when organizations consider new software investments. Sometimes software adheres to one common standard, enabling user organizations to choose among competing packages based on features such as price, performance,

usability, etc. Most often, however, compatibility is not a clear binary issue. As standards and packages evolve and producers compete against each other, packages may converge or diverge on some features, such as, reaching or breaking compatibility.³³ Of course, this development can be caused by legitimate technical design and implementation decisions, but it may also be caused by the producer's perceived advantage in changing the degree of compatibility or interoperability with competing packages.

A producer may differentiate its package from the competition by adding proprietary features and unwarranted proprietary extensions to an open standard. There are some calls for the execution of this predatory business technique of "embrace, extend, and extinguish," and often Microsoft is associated with an almost flawless execution of the technique. Only the law suits that doubtlessly follow spoil the perfection. One historical example is the fight between Sun and Microsoft over Java and extensions to Java.³² The practice of adding proprietary extensions to an (open) standard is successful when some adopters find the proprietary features attractive and implement them. However, it is important to be aware that proprietary features that might be useful for the singular adopter are in fact false gold for the network at large. Every time a proprietary feature is implemented it adds to the switching costs, meaning that it will be harder to pull away from the software package that embeds the proprietary extensions.³⁴ For the network, it means that proprietary features become entrenched as de facto standards, and for the community in general, it becomes an insurmountable barrier to change, thus diminishing the value of a standard.

The break-down of open standards happens in many cases where there is no central governance of a standard by a central institution or authority, and even if such governance does exist, standards often break down anyway as competitors extend the limits of the standard.⁹ One example that we claim to be false gold comes from the company Linksys (owned by Cisco) which has extended its wireless network equipment with proprietary protocols, thus doubling the throughput of the non-propri-

a By false gold or fool's gold we mean something that appears attractive but in reality is not valuable at all.

etary protocol IEEE 802.11b. While the products are still backward compatible with the open standard backed by the IEEE, Linksys gives users a strong incentive to use Linksys hardware exclusively. Another large manufacturer, D-Link, does exactly the same thing; however, the proprietary extensions of D-Link and Linksys are not compatible. For the community, the danger of proprietary extensions is that it may not be compatible with the next generation of the open standards (in this case IEEE 802.11n), and if the proprietary extensions have become entrenched, none is willing to adopt the next open standard version. Thus, the network has moved from a situation where organizations could choose to buy open standard compatible equipment from a number of independent suppliers to a situation where standard evolution has stopped and there is only one supplier of a proprietary de facto standard. In fairness, it should be noted that neither D-Link nor Linksys has been successful in their effort to manifest their proprietary extensions as de facto standards; however, the risk remains.

Organizations should keep their options open by buying packaged software that is close to compatible standards; and if they are already using proprietary standard packages, they should keep their eyes open for gateway standards as a way to break an existing lock-in to a proprietary extension.¹³ At the very least, organizations should be conscious of the adoption of proprietary extensions, document their use in the organization, and consider which steps will be necessary to discontinue their use in the future; that is, a viable exit strategy.

Generic software packages do not meet all the requirements of an organization;^{8,28} there are therefore plenty of options offered as part of the package to configure it as needed.^{16,30} Often local practices or cultural issues add to the desire to customize or localize the package.^{5,22} Customization is different from configuration in that customization is more radical and adds functionality that was not an intended generic feature in the original package. Customization is more lucrative for local software vendors compared to selling the package itself. For the adopting organization, the option to customize may appear shiny,

but for several reasons, could turn out to be false gold.^{20,23,35} First of all, the customization is often expensive and represents sunk costs that, in practice, limit the choices when the package or service contract is up for renewal.^{5,20} Second, when upgrading the software to the next version, usually all customizations have to be re-implemented. In addition, the new features of the next version are obviously not part of the customization that was implemented from the previous version.⁸ Beatty and Williams⁵ recommend “un-customizing customizations” before any upgrade is attempted because they are found to form major technical obstacles and are the main threat to achieving a *Return on Investment*. Instead, Beatty and Williams⁵ propose that an upgrade is an opportunity to review critically existing customizations in order to determine whether they are really needed, and if so, to determine if they are supported in the new version and eligible for elimination. In line with this, we advocate avoiding any comprehensive customization of packaged software, unless absolutely necessary.

Principle Five: Choose a software package with accessible knowledge.

When an organization chooses to use custom-built software, it must carry the entire burden of training and retaining personnel to develop the necessary skills to use the software. The use of packages, however, promises access to knowledge of the package’s application and implementation. Ideally, the network of organizations using a package is matched by a network of individuals competent in configuring and using it, but often the supply and demand of certain skills is not aligned, as is pointed out by Light.²³ If there is an unmet demand for knowledge and skills, both user and producer organizations suffer. One historical example of misaligned networks is that of ERP systems, where the number of people with knowledge and skills of the configuration of SAP systems is far less than the demand from user organizations. The result is disproportionately high costs for the people component of SAP implementations and delayed projects with reduced or poor functionality.

Producers employ various strategies for ensuring a pool of knowledgeable users for their software.¹¹ One strategy is to produce free or low cost versions so that interested people will be more likely to sample it. Another variation is to make “academic versions” of the software package available as free downloads, or to bundle the package with textbooks used in educational institutions. The process of institutionalizing skills is more complex for packages based on open source (sendmail, emacs, Linux, among others), where there may be no single trusted certifying institution corresponding to the owner or vendor of a package. Instead, other forms of legitimization are used, such as a person’s rank in recommender-systems such as discussion Web sites. Such online networks also make it possible to determine the contributions of a particular member, enabling potential employers to retrieve an account of a person’s skills in regard to a particular software package.

The co-development of the two networks (that of the producers and that of the users) has high path dependence to the point of being quasi irreversible.¹¹ For a new competing software package that starts with essentially no network; the existing network forms a formidable entry barrier that is difficult to break.⁶ If the new package is proprietary and the owners are willing to invest, one way for the new standard package to achieve a critical mass of users is for the owner to bear some or all of the costs for the organizations willing to switch.³³ An alternative approach is to invest in building gateway features into the new standard package, thus easing the transition from an incumbent package.¹³ When Microsoft Word was winning over the majority of the word processing market from WordPerfect in the first half of the 1990s, Microsoft sought to circumvent the knowledge barriers by providing WordPerfect users an easy passage. Microsoft Word featured two gateways: an alternative user interface where Microsoft Word could be made to emulate the keyboard shortcuts of WordPerfect, and “Help for WordPerfect users” where the use of Microsoft Word was explained in terms that WordPerfect

users were accustomed to. We suggest using this principle to assess the available knowledge base for the software package.

Principle Six:

Choose packaged software with the right type of standardization.

Standardization can be achieved at various levels and in many forms in packaged software. Here, we provide an overview of the most common types of standardization because it is important to choose the type that is right for the particular organization, according to its available resources and constraints.

Standardization of user interface is a common strategy employed to limit the need for user training. After some experimental implementations of information systems of a particular type, a dominant design typically emerges, resulting in striking similarities of user interfaces among different software systems. Referring to Web site design guru, Nielsen,²⁷ users spend most of their time on other sites, and therefore prefer new Web sites to be designed similar to the sites with which they are familiar. Dominant designs sometimes become static and end up as anachronisms when the surroundings change. For example, the diskette icon featured in most software applications invokes the “save” function, even though no files are ever saved to diskettes and personal computers no longer have disk drives.

In *standardization of output*, the software package’s only compatibility restraint is that it must produce an output that can be used by recipient users or software. One example is that of Web page production, where different departments in an organization may use very different production techniques as long as their Web pages satisfy agreed-upon requirements. This standardization strategy has the strength of allowing users greater freedom to optimize and personalize their production methods. The strategy also has serious drawbacks if the users ever need to share intermediate data; we would thus not recommend this strategy for most organizational standardization issues.

An organization might choose *standardization of data structure* for one of

By choosing an open standard, an organization can usually choose between numerous compatible software packages, thus bringing the simple advantage of choice.

two reasons: seeking backward compatibility with data stored in legacy systems, or seeking to ensure access to the data from other information systems in the future; that is, forward compatibility. By choosing an open standard, an organization can usually choose between numerous compatible software packages, thus bringing the simple advantage of choice. The disadvantage is that the user organization must abstain from using any proprietary features or extensions of the packages chosen (the false gold mentioned in Principle Four) in order to maintain strict data standardization. Examples of data standards with wide vendor support are the all-purpose information formatting languages XML and the database query language SQL, although both are also subject to standard deviations among the implementations from various producers.

More advanced modes of standardization of data interfaces include *interconnectivity* and *interoperability*.⁴ Interoperable information systems are able to communicate during the execution of a particular task. An everyday example is that the functionality of an electronic spreadsheet program can be employed by a word processing program to perform a calculation inside a text document. More advanced implementations allow interoperability between software running on separate computers - even in different locations or organizations such as most Web services organized in serviced oriented architectures (SOA).¹⁴ Features such as these will have far-reaching implications for the implementation of standard software packages and inter-organizational information systems in the coming years.

Organizations may choose *standardization of skills* by employing only people with a particular education or skill set, or if necessary, to carry the cost of training new employees to some formalized level of training (see Principle Five). Organizations can choose to standardize two types of skills: generic or specific skills. Generic skills are skills that are acquired through education, such as critical thinking, programming, business knowledge, and so on. Specific skills encompass a user’s qualifications with a particular software package, and these may

be certified by the product's producer or a trusted third party (see Principle Five). Every major vendor in the packaged software market has such certification programs, and many are even updated on a continual basis, forcing certificate holders to take new exams in order to preserve their status.

One might argue that if all are using the same standard software package, where does competitive advantage come from? As a rule of thumb, we recommend organizations to follow and standardize in all non-core areas to bring down costs, and in order to differentiate themselves, organizations must be prepared to lead (be an early adopter) and tolerate a higher degree of standard uncertainty in core areas. We will return to the issue of competitive advantage in the conclusion.

Principle Seven:
All journeys start with a first step.

In a market of fast update cycles and many options, some buyers may assume a wait-and-see position, while they let the rest of the market test out competing products, determine the necessary feature sets, and so on.^{3,20} Of course, this strategy will mitigate the risks of investing time and money in a software package which later loses in the market, but we advise organizations not to fall into the wait-and-see trap for the following two reasons. First, a winner will only emerge when organizations actually buy software, so an organization stands a greater chance of finding software that fits its needs if it plays an active role in the selection process (invest in the package).

Second, the further development of packages is inevitable, and thus it is very likely that while an organization is waiting for a package to appear in the marketplace for a perfect fit, its requirements may have changed. In fact, it may never be possible to find a perfect match.³⁶ After a prolonged sampling process and the organization finally selecting a software package, activities such as conversion of legacy data may turn into considerable tasks, as there may be no personnel with expertise in both the legacy system and the new software package.²⁰ Therefore, the best strategy to ensure that a better package is there tomorrow is to

We promote a view of buying software as a continuous process of constantly trying to match available packages with a base of already installed information systems, while anticipating future organizational needs and advantages in technology.

adopt its predecessor today by joining its network. Being part of the network will also ensure that special needs are noted and incorporated into the next version of the package.

Conclusion

Software packages are replacing custom built software at a fast pace. Yet, there is little available advice on how to evaluate and choose among the offered packages. This article highlights seven principles that are related to selecting and assessing software packages. The principles extend beyond the two obvious but narrow factors of price and immediate features, to include a wider networked and multilateral view of software packages. We promote a view of buying software as a continuous process of constantly trying to match available packages with a base of already installed information systems, while anticipating future organizational needs and advantages in technology. Companies should seek to select the package that fits their situation. However, this is not a unilateral decision, as other companies' actions also contribute to the destiny of the package. Software packages are networked and built around standards that allow (and disallow) connection to other software systems and these considerations must be added to the equation, too. It is therefore necessary to adopt a multilateral approach that asserts the benefits of participation from as many parties as possible in the selection process.

The proposed principles are useful in several ways. First, they form a reference point for IT managers when engaging in software acquisition. Second, without the principles, IT managers would have to spend much time condensing these foundations from available theoretical and empirical sources. Third, the principles help IT managers ensure that vital aspects of the software package acquisition process have not been left out or neglected. Finally, the set of principles is an invitation to formulate a disagreement and start a discussion on what constitutes sound software acquisition practices.

Here is a checklist that IT managers can consider in addition to the usual technical features and price, when evaluating a software package purchase:

- What companies are involved in

producing the package?

- ▶ How many companies are already using the package?

- ▶ How many software companies can configure the package?

- ▶ What is the history of the package?

- ▶ Is the package built upon open standards?

- ▶ How is the fit with other packages?

- ▶ What kind of standardization does the package represent?

- ▶ Is customization of the package necessary?

- ▶ Is there an accessible knowledge base for the implementation and exploitation of the package?

- ▶ What are the costs of switching to an alternative package?

- ▶ What are the implications of postponing a decision to adopt?

In what direction is the package evolving? And is our company headed the same way?

The principles can be used prior to making an investment and be used to monitor the vitality of existing packages. To illustrate, when a university built a new campus building it came with a free proprietary facility management system with the new building already encoded. However, using the seven principles, the university management decided that even though the package itself was free of charge, the supporting network around the package was too local and too small for the university to invest in encoding the remainder of its buildings into the package.

Another example of the application of the principles was the company in the field study mentioned earlier. The company used the principles to annually monitor the decision to stay with a package that had been dominant but was losing market share. The question was straightforward: Was the network of users around the software package sufficiently large to provide the package owner with revenue that allowed it to invest in developing the package? For a number of years the answer was positive, but when the network was deemed inadequate, it was decided to switch to the dominant package.¹⁷ An illustration of Principle Five and Six is as follows: One large manufacturer had already implemented one ERP system when a vendor offered a competing ERP system at a very competitive price. The manufacturer attempted to switch

but after more than a year of attempting to implement the new ERP system the manufacturer had to revert to its old ERP system. The skill set and knowledge base built around the former ERP system in practice inhibited a switch.

Returning to the competitive advantage discussion initiated earlier and playing the devil's advocate, one might argue that if everybody were using the same software packages, where would competitive advantage in the form of differentiation come from? Succinctly put as a paradox, "In the world of software packages, advantage comes from having the same packages as everybody else before they do." Thus, competitive advantage is gained from being able to spot and adopt the packages of the future before they have become the de facto standard packages, and to identify and phase out the packages of the past before they become legacy systems. ■

This research was in part supported by the Danish Research Foundation, grant number 331958.

References

- Adam, A. and Light, B. Selling packaged software: An ethical analysis. In *Proceedings of the 12th European Conference on Information Systems*. (Turku, Finland, 2004).
- Attewell, P. Technology diffusion and organizational learning: The case of business computing. *Organization Science* 3, 1 (1992), 1–19.
- Au, Y.A. and Kauffman, R.J. Should we wait? Network externalities and electronic billing adoption. *Hawaii International Conference on System Sciences*. (Hawaii, 2001).
- Bailey, J., McKnight, L., and Bosco, P. The economics of advanced services in an open communications infrastructure: Transaction costs, production costs, and network externalities. *Information Infrastructure and Policy* 4 (1995), 255–277.
- Beatty, R.C. and Williams, C.D. ERP II: Best practices for successfully implementing an ERP upgrade. *Commun. ACM* 49 (Mar. 2006), 105–109.
- Besen, S.M. and Farrell, J. Choosing how to compete: Strategies and tactics in standardization. *Journal of Economic Perspectives* 8, 2 (1994), 117–131.
- Chapman, M.R. *In Search of Stupidity: Over 20 Years of High-Tech Marketing Disasters*. Apress, Berkeley, CA, 2003.
- Chlason, M.W. and Green, L.W. Questioning the IT artefact: User practices that can, could, and cannot be supported in packaged-software designs. *European Journal of Information Systems* 16, 5 (2007), 542–554.
- Damsgaard, J. and Lyytinen, K. The role of intermediating institutions in diffusion of electronic data interchange: How industry associations in the grocery sector intervened in Hong Kong, Finland, and Denmark. *The Information Society* 17, 3 (2001), 195–210.
- David, J.S., Schuff, D., and Louis, R.S. Managing your total IT cost of ownership. *Commun. ACM* 45, 1 (2002), 101–106.
- David, P.A. Clio and the economics of Qwerty. *The American Economic Review* 75, 2 (1985), 332–337.
- David, P.A. Narrow Windows, Blind Giants, and Angry Orphans: The Dynamics of Systems Rivalries and Dilemmas of Technology Policy. Technological Innovation Project (No. 10), Stanford University, CA, 1986.
- David, P.A., and Bunn, J.A. The economics of Gateway technologies and network Evolution: Lessons from electricity supply history. *Information Economics and Policy* 3 (1988), 165–202.
- Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, Upper Saddle River, NJ, 2005.
- George, J.F. (ed.) *The Origins of Software: Acquiring Systems at the End of the Century, Framing the Domains of IT Management: Projecting the Future through the Past*. Pinnaflex Educational Resources, Inc., Cincinnati, OH, 2000.
- Howcroft, D. and Light, B. Reflections on issues of power in packaged software selection. *Information Systems Journal* 16, 3 (2006), 215–235.
- Karlsbjerg, J. Staying outside the mainstream: An empirical study of standards choices. *Hawaii International Conference on System Sciences*, Hawaii, 2002.
- Karlsbjerg, J., Damsgaard, J., and Scheepers, R. A taxonomy of Intranet Implementation strategies: To make or to buy? *Journal of Global Information Management* 11, 3 (2003), 151–165.
- Klein, H.K. and Myers, M.D. Evaluating interpretive field studies. *MISQ* 23, 1, 67–94.
- Khoo, H.M. and Robey, D. Deciding to upgrade packaged software: A comparative case study of motives, contingencies and dependencies. *European Journal of Information Systems* 16, 5 (2007), 555–567.
- Klein, B. 1998. Microsoft's use of zero price bundling to fight the browser wars. *The Progress & Freedom Foundation*. Kluwer Academic Publishers, Washington, DC, 1998.
- Kutar, M. and Light, B. Exploring cultural issues in the packaged software industry: A usability perspective. In *Proceedings of the 13th European Conference on Information Systems* (Regensburg, Germany, 2005).
- Light, B. Potential pitfalls in packaged software adoption. *Commun. ACM* 48, 5 (May 2005), 119–121.
- Light, B. and Sawyer, S. Locating packaged software in information systems research. *European Journal of Information Systems* 16, 5 (2007), 527–530.
- Ljungberg, J. Open source movements as a model for organizing. *European Journal of Information Systems*. (Dec. 2000).
- Moore, G.C. End user computing and office automation: A diffusion of innovations perspective. *Infor* 25 (1987), 214–222.
- Nielsen, J. User interface directions for the Web. *Commun. ACM* 42, 1 (Jan. 1999), 65–72.
- Pollock, N., Williams, R., and Procter, R. Fitting standard software packages to non-standard organizations: The 'biography' of an enterprise-wide system. *Technology Analysis and Strategic Management* 15, 3 (2003), 317–332.
- Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1997.
- Sawyer, S. Effects of conflict on packaged software development team performance. *Information Systems Journal* 11, 2 (2001), 155–178.
- Sawyer, S. Information systems development: A market-oriented perspective. *Commun. ACM* 44, 11 (Nov. 2001), 97–102.
- Shankland, S., Kanellios, M. and Wong, W. Sun and Microsoft settle Java suit. *News.com*, 2001.
- Shapiro, C. and Varian, H.R. The art of standards wars. *California Management Review* 41, 2 (1999), 8–32.
- Shapiro, C. and Varian, H.R. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, MA, 1999.
- Sia, S.K. and Soh, C. An assessment of package-organization misalignment: Institutional and ontological structures. *European Journal of Information Systems* 16, 5 (2007), 568–583.
- Truex, D.P., Baskerville, R., and Klein, H. Growing systems in emergent organizations. *Commun. ACM* 42, 8 (Aug. 1999), 117–123.
- West, J. The economic realities of open standards: Black, white and many shades of gray. *Standards and Public Policy*. S. Greenstein and V. Stango (eds.). Cambridge University Press, Cambridge, MA, 2007.

Jan Damsgaard (jd.caict@cbs.dk) is director and a professor at the Center for Applied ICT, Copenhagen Business School, Denmark School of Information Systems, Curtin University of Technology, Australia.

Jan Karlsbjerg (jan@jankarlsbjerg.com) is an associate product manager at Active Community Solutions, Vancouver, Canada.

© 2010 ACM 0001-0782/10/0800 \$10.00