Open Source Enterprise Systems: Towards a Viable Alternative

Alexander Dreiling, Helmut Klaus, Michael Rosemann, Boris Wyssusek
Queensland University of Technology, Australia
{a.dreiling | h.klaus | m.rosemann | b.wyssusek}@qut.edu.au

Abstract

Enterprise systems are located within the antinomy of appearing as generic product, while being means of multiple integrations for the user through configuration and customisation. Technological and organisational integrations are defined by architectures and standardised interfaces. Until recently, technological integration of enterprise systems has been supported largely by monolithic architectures that were designed, and maintained by the respective developers. From a technical perspective, this approach had been challenged by the suggestion of component-based enterprise systems that would allow for a more user-focused system through strict modularisation. Lately, the product nature of software as proprietary item has been questioned through the rapid increase of open source programs that are being used in business computing in general, and also within the overall portfolio that makes up enterprise systems. This suggests the potential for altered technological and commercial constellations for the design of enterprise systems, which are presented in different scenarios. The technological and commercial decomposition of enterprise software and systems may also address some concerns emerging from the users' experience of those systems, and which may have arisen from their proprietary or product nature.

1. Introduction

Preliminary remarks: This paper is about the future of enterprise systems. Consequently, we assume that there is a future for enterprise systems and do not question the concept of enterprise systems in general, although we are aware of some substantial criticism [e.g., 1–3].

The argument of the paper is structured as follows: Reflecting upon the historical development of integrated enterprise information systems we identify the proprietary paradigm of contemporary enterprise systems development as a major obstacle that prevents enterprise systems from ever meeting the promisses made by their proponents as well as meeting the expectations of the companies implementing these kinds of systems. Looking for an alternative to the proprietary paradigm we identify the paradigm of open source software development. After outlining its distinct features we argue—while drawing on various successful open source development initiatives—that open source software development provides us with a viable alternative to the proprietary development of enterprise systems. In our closing discussion we point to some limitations of our exploratory study. We focus on neglected aspects that question some fundamental assumptions underlying the concept of enterprise systems.

2. Enterprise system integration — architecture between developer and user

Enterprise system software is being written by large-scale developers and licensed and implemented by consultants. Despite being called software, these packages have nothing to do with 'shrink-wrapped', 'off-the-shelf' items that can be used instantaneously, rather "[implementing] complex packages such as those offered by SAP, for example, requires a tremendous configuration and customization effort" [4], and "involves the development and utilization of large amounts of specialist knowledge and expertise" [5]. In other words, offerings of software tagged as enterprise resource planning (ERP) or enterprise systems (ES) and regarded as 'solution', i.e., as an item with pre-specified attributes that holds a potential of improvement of business operations and management, are "commodified forms of technical knowledge" [6].

The distinct characteristic of enterprise systems is therefore inseparable from the transactions between developers and user organisations. This suggests that the commodity form of large-scale computer applications should be accounted for, when their 'match' with 'business needs' is contemplated, which means nothing more and nothing less than that enterprise systems packages contain their own rationale against which they are assessed by the purchasing organisation, excluding simultaneously any other criteria to judge its promoted capabilities [3]. Even, when accepting that logic, the complaints about the shortcomings of those systems seem to be endless.

One of the capabilities that remains a contentious issue is the leading concept of enterprise systems: *integration*—which had already been the "holy grail of MIS" since the 1960s [4, 7], when the vision of the totally integrated information system for the whole corporation was put forward by Stafford Beer [8].

On the level of software, integration has meant for a long time that enterprise system software by nature had a "monolithic architecture" [7, 9]. It could be argued that the innovation of enterprise systems is just that, namely an architecture that provides information processing functionality as well as business functionality through 'inter-functional' or process support. In other words, enterprise systems architectures integrate various subsystems, subsumed under the objectives of the entire enterprise.

As commercial products, enterprise systems have striven to cover the requirements of different users within one and the same environment. Developers dealt with the problem of variety and complexity of user organisations by designing software that could be adapted through configurations, with the consequence that ever more options and parameters entered the code. Thus, through incremental evolution, specialised applications, such as Finance, Human Resources, or Logistics, became large applications, called modules [9]. Integration became an issue in that interfaces had to be developed that made possible to combine enterprise systems software with in-house and legacy applications, and add-ons by third-party developers (ibid). However, it might have been against the interests of the large developers to offer truly open interfaces that would allow for combining their products with that of a competitor; third-party developers overcame these obstacles by designing enterprise application integration software [9].

The integration dilemma and the perceived imposition of developers' business models on users [10] occasioned the proposal of an "alternate minimalist strategy … [that] relies on composing large systems from largely independent components that are assembled to meet situation-specific requirements" [7]. The new software 'paradigms' of object orientation and distributed computing, combined with concern for user organisations' 'actual' requirements provided the arguments, or 'technology and business imperatives' for a reversal of the architectural, and necessarily commercial conditions of enterprise computing applications [11].

The architecture that should deliver the desired fit between technology and business, firstly meant changing the current fundamentals in terms of hardware: client-server technology was supposed to be supplanted by distributed object architecture [11]. Next, the monolithic modules were supposed to be abandoned in favour of components that covered specialised processes, and were held to be synonymous with high adaptability, both in terms of current and future special requirements. Components were envisaged as entities that could be assembled with ease into applications. The integration of components within the distributed system was to be accomplished by middleware, as logical layer between components, providing the coordination between them. Component development should become the domain of specialised developers that were close to their clients and the processes in their industries [11].

The accomplishment of the distributed architecture, both in terms of software as well as in terms of the software industry structure, relied heavily on standardisation that ensured interoperability of components and middleware [11]. This kind of integration spawned a plethora of standards, categorised as component interface protocols, integration standards, and semantic agreements [9]. Component interface protocols were CORBA, COM and Enterprise Java Beans, examples of integration standards were XML and EDIFACT, while Microsoft's BizTalk stands for a schema for using XML.

The vision for a new kind of enterprise systems was based on the premise that "[the] monopoly model where a company provides a bundled, monolithic package will not work in the future" [11]. It would have entailed a swift re-orientation of the computer and services industries, including hardware manufacturers. The promise of the "component model of software engineering […] introducing the market-mechanism to software engineering", and by that providing "better products, and lower costs [… and] breaking large monopolies and generating high efficiencies" (ibid), remained unfulfilled from today's perspective. The (relative) openness of application interfaces, since being a strategy by large developers to increase usage of their products, was certainly not a suitable means to shake-up the structure of an industry that is dominated by a handful of 'global players'. The latter could also control either through cooperation or through rivalry the development of the crucial standards that would have allowed or disallowed software engineering principles to be mirrored in industry structure. For users the implications of a turning from big packages from big developers to components from a diverse market would have been a strengthening of the internal IT-expertise, as informed purchaser and wise manager of a tailored portfolio of applications [7]. However, the increased interest in outsourcing, not only information technology, but also business processes, has reduced this expectation to another ambition unfulfilled.

The developments of the past five years thus give ample indication that integration in enterprise systems appears to be only viable by accommodating with the circumstance that systems development remains in the hands of a few 'global players'. Standards that supposedly open development by ensuring interoperability tend to be interpreted by those 'global players' according to their interest. This might be incongruent with the interests of the software industry at large, those of the user organsations, and may also have effects on local and national economies. And, despite control of interfaces and standards by few software developers, even integration of the information infrastructure of one single company with

one brand of enterprise system cannot be consolidated over time [12–16]. In short, managing complexity from a central control perspective has too many trade-offs to be acceptable for good.

We may conclude that software engineering principles and open standards are a necessary but not sufficient condition for enterprise software development becoming less constrained by the politics of 'global players', responsive to user interests, and for ensuring a healthy software industry that can cater for regional markets.

In the 1950s and 1960s, the entire product of business computing could be controlled by one company: Hardware and software used to be bought and experts used to be hired—all en bloc—from vendors such as IBM. Software became a commodity on its own only after IBM's monopoly power had been legally challenged, and software and hardware had to be unbundled [17]. Subsequently, diversity (and incompatibility) of hard- and software prevailed for some time, yet this situation was soon superseded by large-scale developers controlling architectures [18], while taking advantage of some common standards, e.g., internet protocols. Controlling architectures by means of proprietary software and open standards in the enterprise application industry appears to actually preclude innovation that could be of benefit for many users of enterprise systems.

As the first step of commodifying software was a legal matter, attempting to dissolve the proprietary nature of contemporary enterprise systems development may open an opportunity to level the 'playing field' again, and by that seek to address some deficiencies of enterprise systems. A successful example of the abandonment of proprietary software development is *open source* development. Thus, in the remainder of the paper we will outline major differences between proprietary 'closed source' and open source software development and look at possible consequences of a change in the legal status of enterprise systems software.

3. Open Source vs. 'Closed Source'

In the early days of computing software was not a commodity and developers readily shared the source code of the programs they developed. It was only after IBM was forced to unbundle hardware and software that the latter became a subject of property rights and turned into a commodity. With the source code no longer available to the public, commercial software became 'closed source' software.

3.1. Linux and beyond

Today, Linux is the best known example of open source software development. After initially being considered as immature product of immature nerds, Linux has established itself in the marketplace—faster and more successful than anyone ever has anticipated. Some illustrative examples: the biggest computer hardware manufacture sells its computer hardware with Linux installed, provides professional support and actively participates in the further development of Linux; the largest enterprise systems developer has ported its enterprise systems software so that it can be installed ontop of a Linux platform; the city of Munich is about to switch 15.000 users from Microsoft Windows to Linux; the French government contemplates switching about 500.000 of its computers from Microsoft Windows to Linux. See your daily newspaper for ever more examples.

Yet the success of open source software development is not confined to the operating system Linux. In a recently revised online article—"Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!"—David Wheeler provides us with a comprehensive overview on open source software development outcomes [19]: the Apache web server has a market share of about 67 percent; Sendmail is the leading e-mail server with a market share of above 40 percent; more than 75 percent of domain name server (DNS) are serviced by BIND; OpenSSH, a secure shell application, has a market share well above 65 percent. But it is not only the impressive market share of some open source products that illustrates the viability of open source software development. Wheeler also provides us with data regarding reliability, performance, scalability, security, and total cost of ownership—all in favour of open source software products. Besides these quantitative data, Wheeler also outlines a number of qualitative issues such as freedom of control by single source, freedom from licensing management, greater flexibility, and encouraging innovation (ibid)—again, all in favour of open source software products.

Despite the success and the popularity of open source software (development), the common understanding of "open source" does barely reflect its very nature. Yet this is true for the termini "proprietary software" and "free software" as well. In the next section we will highlight some distinct features that can be used to differentiate between major *categories of software* and paradigms of software development.

3.2. Categories of software: free vs. non-free, open source vs. 'closed source'

Open source and 'closed source' software are commonly differentiated based on the belief that the former comes *free of charge* whereas the latter does not. Actually, free in the context of open source rather refers

primarily to its legal status than to the fact that a charge is asked for its use.

Following the definition of the Free Software Foundation, "free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software" [20]. Consequently, "free" does not preclude that free software or open source software can be commercial products. Software that comes free of charge may still be protected by intellectual property rights, and thus not be free—in the sense of the Free Software Foundation. Similarly, the general availability of source code does not imply that it can be used by everyone, and hence is no criterion for "free" either.

If we draw a distinction between open source and 'closed source' (proprietary) software we address certain features of the software development process. These features were outlined by Eric S. Raymond [21]. He contrasts the open source software development model with the classical model of proprietary software development, referring to the former as "bazaar" and to the latter as "cathedral". The bazaar model is characterized by the openness of the source code which allows basically everyone to participate in the development of open source software. The cathedral model is characterized by a limited number of people having exclusive access to the software under development (ibid). Therefore we speak about "closed source". Yet, even if the general public has access to the source code of proprietary software, it does not mean that it is free, since the public does not have the right to "run, copy, distribute, study, change and improve the software" [20]. In short, software is only open source software, if it is also free (FSF definition) software. Again, this does not preclude that open source software can be commercial software.

Due to its distinct characteristics, open source software development (OSSD) has major consequences on the outcome of the development process. Some examples:

- OSSD supports the development of open standards. Based on open standards it is possible for other parties to provide—even proprietary—extensions to existing software.
- OSSD implies open data formats. Users can access data without being bound to proprietary software that may not be available in the future.
- OSSD supports customizability. Users can modify open source software in order to make it meet their specific requirements. One major issue with respect to customizability is *localization*. Since most proprietary software has been developed in order to make profit, its features address only the needs of profitable markets.
- OSSD supports improved quality. Since the development of the source code is not bound to a limited group of developers it becomes easier to detect bugs, conceptual errors, and the like. Also, users can participate in the debugging and (further) development of the software.
- OSSD can help to speed up development processes, since the number of developers involved in an open source software development project is not limited. As we could see from the development of Linux, it is possible to develop even complex systems in a relatively short time.

From the effects of the 'bazaar style' software development process on its outcome, it is easy to derive major deficiencies of the 'cathedral style' software development process. We address some of these deficiencies in the next section, where we use enterprise systems development as an illustrative example.

3.3. Consequences of 'cathedral style' software development for enterprise systems development

Enterprise systems have been touted as the solution to the following problem: "if a company's systems are fragmented, its business is fragmented" [10]. Thus, enterprise systems aim at the integration of all data and data processing functions within the scope of an enterprise, and more recently even beyond [e.g., 22]. The latter may even shift processes within a vertical integration arrangement, such as Vendor Managed Inventory (VMI), where stock replenishment is outsourced to the vendor of an organisation, including demand forecasting [23].

Success stories about enterprise systems report tremendous positive economic effects, e.g., cutting down the time required to fulfil an order, to re-prize products or to complete a credit check—all by huge numbers [e.g., 10]. These gains come at the price of a costly and lengthy implementation process, and, while once in operation, users attribute limited functionality, lack of decision support, lack of extended enterprise support, upgrade difficulties, and high total cost of ownership to their enterprise system infrastructures [22]. Certainly, the blame for these shortcomings can be put easily on the developers and consultants performing the implementation. This can be corroborated by pointing to the acknowledgement by many economists that dominant companies, such as those in the enterprise software industry, are less disposed to respond to articulated customer requirements, and that monopolies as well as oligopolies tend to stifle product and service innovation.

Thus, we want to argue that dissatisfaction with enterprise systems can be explained by the constellation between developers and users, which is in favour of the developers, and that a significant cornerstone for that is the proprietary nature of software licensed to

organisations. Positing the proprietary nature of software as determinant of the dissatisfaction with enterprise systems simply means to shift the attention to the mode of production, rather than being focussed on the point of application. Considering the interests of developers and their control of production allows for recasting client dissatisfaction according to the following examples:

- *Insufficient standardisation of enterprise systems functionality*: Standardisation of enterprise systems functionality would enable easy creation of, e.g., enterprise systems web services and the establishment of enterprise systems web service providers. However, why would an enterprise systems developer be interested in functionality standards if this meant opening his client-base to competition?
- *Insufficient differentiation of technical and business expertise*: Separation of technical expertise (e.g. database technology or workflow technology) from business expertise (e.g. balance sheet creation or payroll) would enable smaller companies to enter the market by offering business expertise for existing technical expertise or the other way around. However, if a large proprietary company offers both in an integrated way and implementing consultants operate accordingly, why would they foster such a development of technology and expertise?
- *Insufficiently opened architecture*: Open architectures would allow for easy integration between business partners. However, if a proprietary enterprise systems developer developed an entirely open architecture, migration would be facilitated and he would make it simpler to replace his product components.
- *Cultural misfit of enterprise resource planning (ERP) packages*: Different cultures certainly need enterprise systems with different functionality. Some of these problems are obvious (Sarbanes-Oxley requirements for SEC-listed companies, BASEL II for company loans in Europe, or US-GAAP for US-compliant balance sheet creation) and even if problems with cultural fit have been reported [24], how can an enterprise systems developer solve a particular issue of a region if he does not operate there?

3.4. Consolidation

The preceding two sections have indicated that open source 'bazaar style' software development can successfully address a number of deficits of contemporary enterprise systems. Neither are these deficiencies of enterprise systems unknown to developers nor are they insurmountable in principle. Rather the problem is the way how software is developed within the constraints of the proprietary 'closed source' model. If the legal status of software developed changes, the unwanted consequences of the 'cathedral style' development process will most likely disappear as well. Thus, transferring enterprise software into the legal status of open source software gives developers the opportunity to address a range of concerns regarding the deficiencies of contemporary enterprise systems. Enterprise systems developed in 'bazaar style,' might also tip the balance of power in favour of the users.

4. Enterprise Systems and Open Source Software Development

Our intention is not to predict the future but rather to contemplate about possibly viable alternatives to the contemporary prevailing mode of enterprise system development. We seek to make these alternatives apprehensible by proposing scenarios that show potential constellations of 'cathedral style' (i.e., proprietary) and 'bazaar style' (i.e., open source) software development for enterprise systems.

4.1. Scenarios including Open Source Enterprise Systems

The decomposition of enterprise systems into an *architecture layer* and a *business application layer* combined with the 'cathedral style' and 'bazaar style' development paradigms leads to four different scenarios of interaction between enterprise systems components (in addition to the status quo of enterprise systems as a monolithic block without evident separation between architecture and business applications). Similar decompositions took place with computer systems into hardware and software, or software into operating systems and application systems. The separation shown in Figure 1 is only one of several possibilities and depicts only a set of possible scenarios including open source enterprise systems. Again, we do not aim to predict the future as to how a separation will take place, but given the fact that enterprise systems developed into what they are today it seems likely that any kind of separation will take place. All scenarios except for the first one require open interfaces between the architecture and business application layers and we argued above that it is highly unlikely that they will emerge under the 'cathedral style' development process.

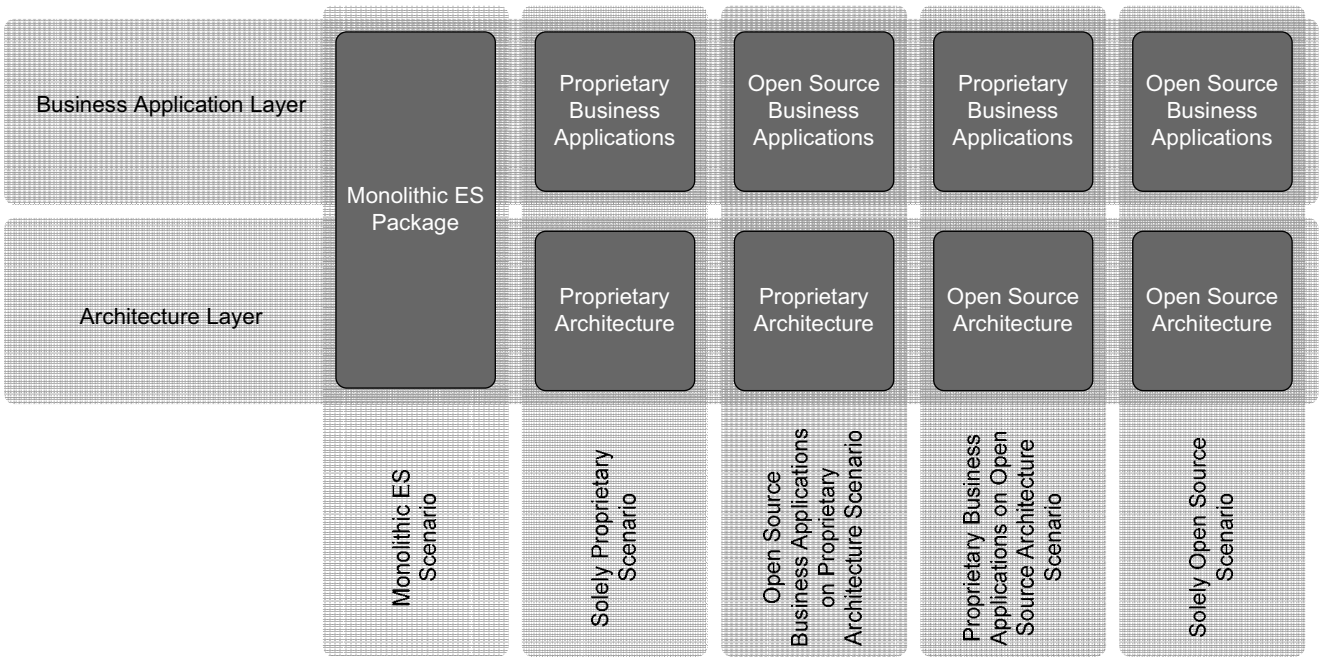

Figure 1: Possible development scenarios based on a decomposition of contemporary enterprise systems including open source components

The *monolithic ES* scenario is the status quo. There are many examples of such enterprise systems on the market and we have elaborated on the implications and consequences of these packages.

The *solely proprietary* scenario separates architecture and business applications. Both component types are proprietary but the architecture does not necessarily have to be open for third parties which would still imply a 'cathedral style'. An enterprise systems vendor (or an alliance of such vendors) may be interested in this clear separation in order to develop different modules more efficiently. A similar separation of architecture from applications outside the enterprise systems domain is Microsoft's separation of operating systems and applications.

The *open source business applications on proprietary architecture* scenario offers a highly standardised open architecture acting as a communication and integration medium between open source components. A popular example for such this constellation of proprietary and open source software would be freeware based on Microsoft operating systems such as Open Office. However, open source software in this constellation is highly depended on the vendor of the proprietary architecture. Control of sophisticated proprietary open architectures in the first scenarios would most probably lead to considerable success for their vendors [18]. However, this success is not the focus of this paper as we explore the effects of a different software development paradigm of enterprise systems.

The *proprietary business applications on open source architecture* scenario were possible if, e.g., an initiative of enterprise systems vendors defined an architecture in order to focus on an efficient provision of business applications. Popular examples for this constellation outside the enterprise systems domain would be proprietary software such as SAP's R/3 on the operating systems Linux. For the argument made in this paper so far, this scenario would be more preferable than the last one, because the market power of a strong vendor of a proprietary architecture cannot be abused and some of the discussed limitations resulting from the proprietary software development paradigm of enterprise systems could be avoided.

The *solely open source* scenario could be in the interest of a party offering services on top of enterprise systems such as implementation services (or a joint initiative of a set of such parties), which in fact is a huge market in itself. Given the specificity of different industries and companies within these industries, it is highly unlikely that enterprise systems especially for large companies will ever be implemented without configuration. Thus an initiative of consulting companies may be interested in jointly developing an open source architecture and open source business applications to sell configuration on top of the software.

5. Conclusions and Outlook

Enterprise systems have been discussed in academia for several years. It appears, as if the perspective of industry analysts is prevailing in considering the development of enterprise systems as an extension of their status quo into the future, based on assumptions how companies can position themselves in relation to their customers and competitors. However, as Gartner Group missed out to predict the industry leadership of SAP, it may be questioned whether this view on the development of enterprise systems was meaningful. We have suggested proceeding rather by questioning whether the enterprise system's underlying integration concept and development paradigm are dimensions that due to their inherent antinomies might be susceptible to change.

Technical possibilities for novel constellations of integration and development are inherent in the architecture for enterprise systems, where the emergent vertical split of enterprise systems into architecture and business applications can be observed. Simultaneously, the open source paradigm suggests a novel approach to development, coexisting with the proprietary one. The vertical split of enterprise systems may show some potential for the extension of open source into that area. Given the central tenets of enterprise systems, we have suggested several reasons for an open source enterprise systems development being preferable over its proprietary counterpart.

In fact, proprietary software has emerged only forty years ago. The product nature of software and its promises have often been unfulfilled in that large developer companies disappeared as quickly as they had boomed. On the other hand, proprietary software can also be held to be a significant factor contributing to the quasi-monopoly of Microsoft on the office systems and operating systems market, with its concomitant severe security threats due to viruses and worms and alleged monopoly abuse in several cases. Proprietary software also raises the barriers for less developed economies to become users of information technology, or to participate in the software industry.

Thus, a range of themes addressed in this paper warrant further investigation. Especially the forms of co-existence of open source and conventional development may yield some further insights into the shifting grounds of systems development and the actions of its 'global players'.

6. References

[1] B. Robinson and F. A. Wilson, "Planning for the market? Enterprise resource planning systems and the contradictions of capital", *The DATA BASE for Advances in Information Systems*, vol. 32, pp. 21–33, 2001.

[2] T. Haigh, "The chromium-plated tabulator: institutionalizing an electronic revolution, 1954–1958", *IEEE Annals of the History of Computing*, vol. 23, pp. 75–104, 2001.

[3] J. Kallinikos, "Deconstructing information packages: organizational and behavioural implications of ERP systems", *Information Technology & People*, vol. 17, pp. 8–30, 2004.

[4] T. Haigh, "Software in the 1960s as concept, service, and product", *IEEE Annals of the History of Computing*, vol. 24, pp. 5–13, 2002.

[5] S. Newell, J. Swan, and R. D. Galliers, "A knowledge-focused perspective on the diffusion and adoption of complex information technologies: the BPR example", *Information Systems Journal*, vol. 10, pp. 239–259, 2000.

[6] H. Scarbrough, "Blackboxes, hostages and prisoners", *Organization Studies*, vol. 16, pp. 991–1019, 1995.

[7] K. Kumar and J. v. Hillegersberg, "ERP experiences and evolution", *Communications of the ACM*, vol. 43, pp. 22–26, 2000.

[8] S. Beer, *Decision and control: the meaning of operational research and management cybernetics*. London: Wiley, 1966.

[9] D. Sprott, "Componentizing the enterprise application packages", *Communications of the ACM*, vol. 43, pp. 63–69, 2000.

[10] T. H. Davenport, "Putting the Enterprise into the Enterprise System", *Harvard Business Review*, vol. 76, pp. 121–131, 1998.

[11] M. Fan, J. Stallaert, and A. B. Whinston, "The adoption and design methodologies of component-based enterprise systems", *European Journal of Information Systems*, vol. 9, pp. 25–35, 2000.

[12] C. U. Ciborra, "Notes on improvisation and time in organizations", *Accounting, Management and Information Technologies*, vol. 9, pp. 77–94, 1999.

[13] C. U. Ciborra and O. Hanseth, "From tool to *Gestell*: agendas for managing the information

infrastructure", *Information Technology & People*, vol. 11, pp. 305–327, 1998.

[14] O. Hanseth and K. Braa, "Who's in control: designers, managers – or technology? Infrastructures at Norsk Hydro", in *From control to drift: the dynamic of corporate information infrastructures*, C. U. Ciborra, Ed. Oxford: Oxford University Press, 2000, pp. 126–147.

[15] O. Hanseth and K. Braa, "Technology as traitor: emergent SAP infrastructure in a global organization", presented at 19th International Conference on Information Systems, Helsinki, 1998.

[16] O. Hanseth, C. U. Ciborra, and K. Braa, "The control devolution: ERP and the side effects of globalization", *ACM SIGMIS Database*, vol. 32, pp. 34–46, 2001.

[17] H. Scarbrough, "Problem-solutions in the management of information systems expertise", *Journal of Management Studies*, vol. 30, pp. 939–955, 1993.

[18] C. R. Morris and C. H. Ferguson, "How Architecture Wins Technology Wars", *Harvard Business Review*, pp. 86–96, 1993.

[19] D. A. Wheeler, "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!" http://www.dwheeler.com/, 2004.

[20] Free Software Foundation, "The Free Software Definition", http://www.gnu.org/, 2004.

[21] E. Raymond, "The Cathedral and the Bazaar", *Knowledge, Technology & Policy*, vol. 12, pp. 23–49, 1999.

[22] M. L. Markus, D. Petrie, and S. Axline, "Bucking the Trends: What the Future May Hold for ERP Packages", *Information Systems Frontiers*, vol. 2, pp. 181–193, 2000.

[23] SAP AG, "Collaborative Business Scenario 'Vendor Managed Inventory'", http://www.sap.com/, 2004.

[24] C. Soh, S. S. Kien, and J. Tay-Yap, "Enterprise resource planning: cultural fits and misfits: is ERP a universal solution?" *Communications of the ACM*, vol. 43, pp. 47–51, 2000.