Journal of Information Technology (2010) 25, 1–19

palgrave-journals.com/jit/

Research article

Design theory for dynamic complexity in information infrastructures: the case of building internet

Ole Hanseth[1], Kalle Lyytinen[2]

[1]Department of Informatics, University of Oslo, Norway;
[2]Department of Information Systems, Weatherhead School of Management, Case Western Reserve University, Cleveland, USA

Correspondence:
O Hanseth, Department of Informatics, University of Oslo, Boks 1072 Blindern, NO-0316 OSLO, Norway.
Tel: +47 95 90 85 09;
Fax: +47 22 85 24 01;
E-mail: ole.hanseth@ifi.uio.no

Abstract

We propose a design theory that tackles dynamic complexity in the design for Information Infrastructures (IIs) defined as a shared, open, heterogeneous and evolving socio-technical system of Information Technology (IT) capabilities. Examples of IIs include the Internet, or industry-wide Electronic Data Interchange (EDI) networks. IIs are recursively composed of other infrastructures, platforms, applications and IT capabilities and controlled by emergent, distributed and episodic forms of control. II's evolutionary dynamics are nonlinear, path dependent and influenced by network effects and unbounded user and designer learning. The proposed theory tackles tensions between two design problems related to the II design: (1) *the bootstrap problem*: IIs need to meet directly early users' needs in order to be initiated; and (2) *the adaptability problem*: local designs need to recognize II's unbounded scale and functional uncertainty. We draw upon Complex Adaptive Systems theory to derive II design rules that address the bootstrap problem by generating early growth through simplicity and usefulness, and the adaptability problem by promoting modular and generative designs. We illustrate these principles by analyzing the history of Internet exegesis.

Journal of Information Technology (2010) **25,** 1–19. doi:10.1057/jit.2009.19
Keywords: design theory; Complex Adaptive Systems; information infrastructure; Internet; historical case study

Introduction

Increased processing power and higher transmission and storage capacity have made it possible to build increasingly integrated and versatile Information Technology (IT) solutions whose complexity has grown dramatically (BCS/RAE, 2004; Hanseth and Ciborra, 2007; Kallinikos, 2007). Complexity can be defined here as the dramatic increase in the number and heterogeneity of included components, relations, and their dynamic and unexpected interactions in IT solutions. Unfortunately, software engineering principles and design methodologies have not scaled up creating a demand for new approaches to better cope with this increased complexity (BCS/RAE, 2004). The growth in complexity has brought to researchers' attention novel mechanisms to cope with it like architectures, modularity or standards (Parnas, 1972; Schmidt and Werle, 1998; Baldwin and Clark, 2000). Another, more recent stream of research has adopted a more holistic, socio-technical and evolutionary approach putting the growth in the combined social and technical complexity at the center of an empirical scrutiny (see, e.g., Edwards *et al*., 2007). These scholars view these complex systems as new types of IT artifacts and denote them with a generic label of *Information Infrastructures* (IIs). So far, empirical studies have garnered significant insights into the evolution of IIs of varying scale, functionality and scope including Internet (Abbate, 1999; Tuomi, 2002), electronic market places and EDI networks (Damsgaard and Lyytinen, 2001; Wigand *et al*., 2006), wireless service infrastructures (Funk, 2002; Yoo *et al*., 2005) or ERP systems (Ciborra *et al*., 2000). At the same time effective design of IIs holds considerable

benefits for individuals, businesses and society at large as testified, for example, by the success of Internet. Yet, failures to design IIs are more common incurring huge losses in foregone investments, opportunity costs, and political and social problems. A case in point is current the difficulty to implement a nation wide e-health system in the UK (Sauer and Willcocks, 2007; Greenhalgh *et al.,* 2008).

One challenge in the II research has been in the difficulty of translating vivid empirical descriptions of IIs evolution into effective *socio-technical* design principles that promote their evolution, growth and complexity coordination. In this paper we make some steps in addressing this challenge by formulating a new design approach to address the *dynamic complexity of IIs*. From a technical view point designing an II involves discovery, implementation, integration, control and coordination of increasingly heterogeneous IT capabilities. Socially, it requires organizing and connecting heterogeneous actors with diverging interests in ways that allow for II growth and evolution. In the proposed approach we posit that the growing complexity of IIs originates from local, persistent and limitless shaping of II's IT capabilities due to the enrollment of diverse communities with new learning and technical opportunities. We argue that one common reason for the experienced II design culprits is that designers cannot design IIs effectively by following traditional top-down design. In particular, the dynamic complexity poses a chicken-egg problem for the would-be II designer that has been largely ignored in the traditional approaches. On one hand, IT capabilities embedded in II gain their value by being used by a large number of users demanding rapid growth in the user base (Shapiro and Varian, 1999). Therefore, II designers have to come up early on with solutions that persuade users to adopt while the user community is non-existent or small. This requires II designers to address head on the needs of the very first users before addressing completeness of their design, or scalability. This can be difficult, however, because II designers must also anticipate the completeness of their designs. This defines the *bootstrap problem of II design*. On the other hand, when the II starts to expand by benefitting from the network effects, it will switch to a period of rapid growth. During this growth, designers need to heed for unforeseen and diverse demands and produce designs that cope technically and socially with these increasingly varying needs. This demands infrastructural flexibility in that the II adapts technically and socially. This defines the *adaptability problem of II design* (Edwards *et al*., 2007). Clearly, these two demands contradict and generate tensions at any point of time in II design (Edwards *et al.,* 2007).

In this paper we will address this tension by examining emergent properties of IIs as *adaptive complex systems.* As IIs exhibit high levels of dynamic complexity, they cannot be designed in the traditional way starting with a 'complete' set of requirements. II designers cannot design IIs just based on the 'local' knowledge, but they can increase the likelihood for successful emergence and growth of IIs by involving elements in their designs that take into account socio-technical *features* of IIs generated by their dynamic complexity. We call this engagement *design for IIs*.[1] While designing *for* IIs, the designers need to ask how they can generate designs that promote continued growth and adaptation of IIs. To this end we outline a socio-technical II design theory (Walls *et al*., 1992, 2004) consisting of design principles and rules (Walls *et al*., 1992, 2004; Baldwin and Clark, 2000; Markus *et al*., 2002). This theory can guide design behaviors in ways that allow IIs grow and adapt *as self-organizing systems*. It is a socio-technical theory, because its design domain involves both technical and social elements and their relationships. It is a design theory, because it consists of 'how to' design principles and rules (Walls *et al*., 1992, 2004; Markus *et al*., 2002) backed by 'because of' justifications derived from a kernel theory – Complex Adaptive Systems (CAS) theory (Holland, 1995). We illustrate the validity of these design principles and rules by following the exegesis of Internet.

The remainder of this essay is organized as follows. In the next section we define IIs and characterize their dynamic complexity. The following section formulates the design theory based on CAS to address dynamic complexity by deriving design principles. In the subsequent section we detail the design rules to address dynamic complexity and illustrate their use during the design of the Internet. In the final section we offer concluding remarks and note some avenues for future research.

Information infrastructures

IT capabilities, applications and platforms

As noted, IIs form a different 'unit'[2] of design when compared with traditional classes of IT solutions. These design classes can be defined in their order of increasing complexity as: (1) IT capabilities, (2) applications, (3) platforms, and (4) IIs. The main differences between these classes lie in their overall complexity, how they relate to their design and use environments, and how they behave over time in relation to those environments. They pose different challenges during the design, are organized differently, controlled differently and obtain distinct emergent properties. The main features of each class are depicted in Table 1.

We denote an *IT capability* as the possibility and/or right of the user or a user community to perform a set of actions on a computational object or process**.** An example of such capability would be a text editor. An IT capability is defined and managed locally by single or a small group of designers. They typically control its evolution locally. IT capabilities are viewed here solely as engineered artifacts.

Applications consist of suites of IT capabilities. They are developed to meet a set of specified user needs within a select set of communities. They can grow amazingly complex in terms of effort and scope, but despite this, they still can be viewed as applications, if governed by a set of specifications[3] through which their design scope remains bounded. An application is *a priori* determined by choice of design context, user groups and functional goals. Consequently, the application can be developed, and preferably should be done so, by a hierarchy assuming centralized control.[4] Therefore, most proposed design theories address the design of applications by promoting ways of generating effectively a closure in the included IT capabilities as to meet user's needs (Boehm, 1976; Ross and Schoman, 1977; DeMarco, 1978; Olle *et al*., 1983; Agresti, 1986; Walls *et al*., 1992; Freeman, 2007).

Table 1 Applications, platforms and information infrastructures

Property/Type of IT system	*Application*	*Platform*	*Information infrastructure*
Emergent properties			
Shared	Yes, locally and through specified functions	Yes, across involved user communities and across a set of IT capabilities	Yes, universally and across multiple IT capabilities (Star and Ruhleder, 1996; Porra, 1999)
Open	No, closed by user group and functionality	Partially, depends on design choices and managerial policies	Yes, universally allowing unlimited connections to user communities and new IT capabilities (Weill and Broadbent, 1998; Kayworth and Sambamurthy, 2000; Freeman, 2007)
Heterogeneous	Yes, partially and mainly by involved social groups	Partially, mainly by social groups but also by technical connections	Yes, increasingly heterogeneous both technically and socially (Kling and Scacchi, 1982; Hughes, 1987; Kling, 1992; Edwards *et al.*, 2007)
Evolving	Yes, but limited by time horizon and user community.	Yes, and limited by architectural choices and functional closure	Yes, unlimited by time or user community (Star and Ruhleder, 1996; Freeman, 2007; Zimmerman, 2007)
	Linear growth	Mostly linear growth	Both linear and nonlinear growth (Hughes, 1987)
	Evolution bounded and context free	Evolution path dependent	Evolution path dependent (Star and Ruhleder, 1996; Porra, 1999; Edwards *et al.*, 2007)
Structural properties			
Organizing principle	Direct composition of IT capabilities within a homogeneous platform	Direct composition of a set of horizontal IT capabilities within a set of homogeneous platforms	Recursive composition of IT capabilities, platforms and infrastructures over time (Star and Ruhleder, 1996; Edwards *et al.*, 2007)
Control	Centralized	Centralized	Distributed and dynamically negotiated (Weill and Broadbent, 1998) Can involve only basisorganizing principles (standards) and rely on installed base inertia (Star and Ruhleder, 1996; Edwards *et al.*, 2007).

Platforms differ from applications due to their heterogeneous and growing user base, that is their design context is not fixed due to the constant generification of included IT capabilities (Williams and Pollock, 2008). Platforms include, for example, office software platforms (MS Office, Officestar), operating system platforms (Windows, Unix), application frameworks like ERP or CRM packages (SAP, Oracle, SalesForge) or application development platforms (e.g. Service Oriented Architecture). Platform designs draw upon architectural principles that organize IT capabilities into frameworks allowing the software to address a family of generic functional specifications that meet the needs of multiple, heterogeneous and growing user communities (Evans *et al*., 2006; Williams and Pollock, 2008). Platforms are composed by formulating a design framework (architecture) that allows organizing a growing set of IT capabilities into a relatively well-bounded and controlled system. The platforms provide thus a (semi)-closed, and highly complex suite of IT capabilities, which, thanks to the original architecting, can be extended. A platform's initial design starts with a set of closed specifications determining included IT capabilities and anticipated requirements for their extensions and combinations. Their evolution is also governed and constrained by these initial specifications. Therefore, the design context remains controlled and the relationships between the user and design communities do not change significantly during the platform's lifetime. Platforms typically grow in complexity as designers take into account heterogeneous user needs while maintaining backward compatibility and horizontal compatibility across different combinations of capabilities. Therefore, many platforms, originally conceived as limited sets of IT capabilities, obtain later emergent features; they start growing in seemingly unlimited fashion and serve unexpected user communities generating exponentially growing technical and social complexity. Consider, for example, the growth of the MS Office platform, or Linux operating system due to the increasingly distributed and open character of their design and user communities (Scacchi, 2009).

Defining II

Based on an extensive literature review we will define next *II*. Hughes (1987) recognized early on heterogeneity, socio-technical nature and unbounded growth as essential features of infrastructures. Kling (1992) and Kling and Scacchi (1982) drew attention to additional material requirements of infrastructures like connectivity. Porra (1999) and Star and Ruhleder (1996) have recently emphasized the criticality of sharing and learning within and across communities, while Kayworth and Sambamurthy (2000), Weill and Broadbent (1998) and Chung *et al.* (2003) have pinpointed the possibility to shape infrastructures by local choice. Finally, recent definitions of IIs recognize tensions between the local and the global, their recursive nature, their unique coordination challenges due to the lack of global control (Star and Ruhleder, 1996; Edwards *et al*., 2007; Freeman, 2007; Zimmerman, 2007) (Table 1).

Accordingly, we will define an *II* as a *shared*, *open* (and unbounded), *heterogeneous* and *evolving* socio-technical system (which we call *installed base*) consisting of a set of IT capabilities and their user, operations and design communities. This definition highlights both the structural properties and the emergent properties of IIs that *distinguish* IIs from their constituent elements (Table 1).

Structurally an II is recursively composed of other infrastructures, platforms, application and IT capabilities. Recursion forms the organizing principle implying that IIs return 'onto' themselves by being composed of similar elements (Lee *et al*., 2006). Socially, IIs are also recursively organized in that they are both outcomes and conditions of design action and involve rule-following and rule-shaping activity (Giddens, 1984). The control of II is distributed and episodic and an outcome of negotiation and shared agreements. Distributed forms of control form often the only way to coordinate II evolution and thus IIs are never changed from above (Star and Ruhleder, 1996). Therefore, they cannot be truly 'designed' in a traditional sense as in traditional approaches a designer assumes control over the design space (Edwards *et al*., 2007; Freeman, 2007). Episodic forms of control determine which groups of designers control which parts or elements of the II; what IT capabilities become integrated and how; who has access to the capabilities and so on (Tuomi, 2002; Edwards *et al*., 2007).

IIs become *shared* across multiple communities in a myriad and unexpected ways. In principle, they exhibit unbounded *openness*: new components can be added and integrated with them in unexpected ways and contexts. In addition, there are no clear boundaries between those that can use an II and those that cannot; and there are no clear boundaries between those that can design the II and those that may not. As a result, II designs need to be approached *as if* no closure, in principle, is assumed in their form or content, capability, form or scope of access.

The openness of IIs implies that during their lifetime the social and technical diversity and *heterogeneity of IIs* will increase (Edwards *et al*., 2007). IIs become increasingly heterogeneous as the number of different kinds of technological components are included, but first of all because IIs include (an increasing number of) components of very different nature: user communities, operators, standardization and governance bodies, design communities, etc.

Finally, because IIs are open, they *evolve*, seemingly, *ad infinitum*. IIs are never built in a green field, nor do they die – though they may wither to rise in new forms (Edwards *et al*., 2007). IIs are often bootstrapped by experimenting and thereby enrolling new communities. For example, Berners-Lee designed the first web service to meet information sharing needs among high energy physicists (Berners-Lee and Fischetti, 1999). As this design unfolded, designers and users discovered additional IT capabilities, or transformed the existing ones to new uses, or to other design contexts thereby expanding the web (Ciborra *et al*., 2000) generating its fractal evolution. Hence, the evolution of 'Infrastructure is fixed in modular increments, not all at once or globally' (Star and Ruhleder, 1996).

Overall, the evolution of infrastructures is both enabled and constrained by the *installed base*,[5] that is the existing configuration of II components (Hughes, 1987; Star and Ruhleder, 1996; Porra, 1999). Whatever is added needs to be integrated and made *compatible* with this base. This sets up demands for horizontal and/or backwards compatibility and imposes constraints on what can be designed at any time. Accordingly, II evolution is path dependent

and shaped by neighboring infrastructures, existing IT capabilities, user and designer learning, cognitive inertia, etc. (Hughes, 1987; Kling, 1992; Star and Ruhleder, 1996; Hanseth and Monteiro, 1997; Porra, 1999).

Related research

Most II research has aimed at identifying the main features and characteristics of IIs – their 'nature' as they evolve and developers and users are struggling to make them work (Star and Ruhleder, 1996; Ciborra *et al*., 2000; Kallinikos, 2004, 2006, 2007; Edwards *et al*., 2007; Contini and Lanzara, 2009).

Standards are core elements of IIs; hence standards research constitutes a major part of II research (Star and Ruhleder, 1996; Edwards *et al*., 2007). A large part of this research has focused on and disclosed a very dense and complex web of relations between technical and social (or non-technical) issues and elements of the standards (Hanseth and Monteiro, 1997; Bowker and Star, 1999; Lyytinen and Fomin, 2002) Another key part of the research has focused on the creation and role of network effects, that is self-reinforcing processes leading to lock-ins (Shapiro and Varian, 1999; Hanseth, 2000).

A minor part of II research has focused explicitly on strategies for developing standards and infrastructures. Cordella (2004), Hanseth and Lundberg (2001), Grisot (2008) and Pipek and Wulf (2009) describe how infrastructures emerge in use while users appropriate a variety of IT capabilities and bring them together in novel ways by making them components of IIs . Even without technically integrating the capabilities, they are *de facto* becoming integrated and interdependent in work practices (Pipek and Wulf, 2009).

A few researchers have addressed design strategies for infrastructure development. Hanseth *et al*. (1996) recognize the need to manage the tension between standardization and flexibility (see also Egyedi, 2002). Hanseth and Aanestad (2003) argue, using telemedicine as an illustration, that II design needs to be seen as a bootstrapping process, which utilizes network effects and spillovers within a growing user base by using simple solutions as a sort of 'stunts,' which offer 'detours' on the road toward infrastructures (Aanestad and Hanseth, 2002). Hanseth (2001) also demonstrates the importance of gateways in flexible II design by reviewing the history of the Internet design in Scandinavia. Lessig (2001) and David (2001) note the criticality of Internet's architectural features – especially its end-to-end architecture – in supporting adaptability at the 'edge of chaos' (Saltzer *et al*., 1984). Benkler (2006) emphasizes the importance of the local 'programmability' of terminals, and its mutual dependency on the end-to-end architecture. Finally, Zittrain (2006) recently combined these features into an encompassing concept of a generative technology – a notion close to our idea of dynamic complexity. We add next to Zittrain's concept a more coherent design framework formulated as a design theory.

Design theory for addressing adaptive complexity in II evolution

IT design theories

Since the publication of Walls *et al*.'s (1992) article, the term 'IS design theory' has denoted a set of concepts, beliefs and 'laws' – either natural or social – which help designers map a class of design problems to effective solutions that meet design goals. Design theories are about 'how to' principles and rules of form and function, and justificatory 'because of' explanatory knowledge that can be mobilized during the design (Gregor, 2006). They encapsulate three elements: (1) a set of design goals shared by a family of design problems; (2) a set of system features that meet those goals; and (3) a set of design principles and rules to guide the design so that a set of system features is selected to meet chosen design goals. Design principles state broad guidelines how the design can be carried out and where the designer can focus his or her attention during function and form shaping. They can be further detailed into design rules that formulate in concrete terms how to generate and select desired system features as to achieve stated system goals.

The crux of a design theory is its 'kernel theory' (Walls *et al*., 1992). It postulates falsifiable predictions for a class of design solutions (i.e. product theories), or design processes (i.e. process theories) in relation to system goals. They vary significantly in their generality, structure and predictive power (Gregor, 2006). Each design theory applies *in a certain design context* in which *a specific set of system goals* have been selected, and *apply to a specific class of systems* and *associated design processes*. The design context is determined by the nature of the system, its size, the design phase, the type of technology, the type of users or designers (Walls *et al*., 1992, 2004). Next we will focus on II design theory for dynamic complexity. Our main interest is in generating technical and social components of the IIs in ways that address the tensions between the bootstrap and the adaptability problems.

CAS as a design theory about dynamic complexity in II

We draw upon CAS theory as our kernel theory (Holland, 1995; Benbya and McKelvey, 2006). CAS addresses nonlinear phenomena within physics and biology, but also in social domains including financial markets (Arthur, 1994). CAS investigates systems that *adapt* and *evolve* while they *self-organize*. The systems are made up of autonomous agents with the ability to adapt according to a set of rules in response to other agent's behaviors and changes in the environment (Holland, 1995). Key characteristics of CAS are: (1) nonlinearity, that is small changes in the input or the initial state can lead to order of magnitude differences in the output or the final state; (2) order emerges from complex interactions; (3) irreversibility of system states, that is, that change is path dependent; and (4) unpredictability of system outcomes (Dooley, 1996).

We chose CAS as our kernel theory as it recognizes factors that generate the dynamics associated with the II bootstrap and adaptability problems and helps describe II evolution as an example of path-dependent and nonlinear change. CAS brings theoretical rigor to generate insights to these two design challenges. In addition, these challenges are not highlighted in two pet theories among II scholars: the social shaping of technology (Edwards *et al*., 2007), and Bateson's ecological theory (Star and Ruhleder, 1996). Next, the design principles are deductively derived from CAS. In the section 'Design rules to manage dynamic complexity – the Internet case' we instantiate these principles with a set

of 19 design rules whose application is illustrated with concrete episodes from the design history of Internet.[6]

CAS categories and design principles for dynamic complexity in IIs

CAS helps characterize how the IIs can be initiated and how they grow and evolve while they self-organize. This is addressed by following the two principles: (1) create an attractor that feeds system growth to address the bootstrap problem; and: (2) assure that the emerging system will remain adaptable at 'the edge of chaos' while it grows to address the adaptability problem. We surmise that these principles can promote design for IIs in ways that lead them to self-organize and to grow. We will next describe key categories of CAS theory and the logic that underpins these design principles and their 'because of reasons' as suggested in Table 2.

Addressing growth in II

A central claim in CAS is that order *emerges* – it is not designed by an omnipotent 'designer.' Typical example is the dynamic arrangements among cells and the establishment of standards without anyone ever intending to design them as such (e.g. QWERTY, TCP/IP). According to CAS, such orders emerge around *attractors*, that is a limited range of states within which the system growth can stabilize, and which allow the system to bootstrap (Holland, 1995). The simplest attractor is a single point in the system state space. Attractors can also come in other forms, which are called 'strange attractors' (Carpa, 1996) that stabilizes the system into a specific region (David, 1986). De-facto standards (e.g. MS Windows, QWERTY, Internet standards) are examples of such attractors. Attractors stabilize a system through *feed-back* loops (also called network effects or 'increasing returns'). In case of standards this happens because the value of a standard defining an IT capability depends on the number of users having adopted it. So when a user adopts a standard its value increases. This again makes it more likely that another user will adopt it, which further increases its value and so on (Arthur, 1994; Shapiro and Varian, 1999). A large installed base will also attract complementary IT capabilities thereby making the original capability increasingly attractive (Shapiro and Varian, 1999). A larger installed base increases also the credibility associated with the capability and reduces user risks of foregone investments. Together these features make an IT capability more attractive leading to increased adoption that further increases its installed base (Grindley, 1995). Some describe this process of getting 'the bandwagon moving.'

Positive network effects lead to self-reinforcing *path-dependent* processes. Overall, the involved path dependency suggests that past events – for example a serendipitous adoption, or correctly timed designs – can change history by generating irreversible effects – called butterfly effects. Such path-dependent growth will eventually lead to a *lock-in* when the adoption rates cross a certain threshold (David, 1986). Such a lock-in happens when a system's growth reaches what Hughes (1987) calls a momentum. This creates a new lasting order with irreversible effects (Arthur, 1994).

We distinguish two facets of path dependence in IIs: *cumulative adoption* and *technology traps*. Cumulative adoption takes place when an II designer builds up an

Table 2 CAS-based design theory for dynamic complexity in Information Infrastructures (IIs)

Design goals	Bootstrap the IT capability into an installed base so that it gains momentum Manage and allow for maximum II adaptability.
A set of system features	II as an unbounded, evolving, shared, heterogeneous and open recursively organized system of IT capabilities whose evolution is enabled and constrained by its installed base and the nature and content of its components and connections.
Kernel theory of flexible IIs	CAS informs how to address *bootstrap problem* in II designs by suggesting that • Designer can gain momentum in the growth of II through attracting a critical mass of users • Designer can enable nonlinear growth by new combinations of the installed base CAS informs how to address the *adaptability problem* in II designs by suggesting that • Designer needs to recognize path dependencies within the installed base • Designer needs to create lock-in through network externalities that exclude alternative pathways • Designer need to achieve modularity to accommodate the growing need for openness and heterogeneity in future
Design principles	For the II bootstrap problem: 1. Design initially for usefulness 2. Draw upon existing installed base 3. Expand installed base by persuasive tactics For the II adaptability problem: 4. Make each IT capability simple 5. Modularize the II by building separately its principal functions and sub-infrastructures using layering and gateways

installed base ahead of its alternatives and accordingly becomes cumulatively attractive, and starts growing in an unbounded manner. Bootstrapping for cumulative growth is possible when the timing is right and users can still be persuaded (Edwards *et al*., 2007; Zimmerman, 2007). Design choices for II thereby become path dependent, as conditions for cumulative adoption are created during the early stages of growth (Grindley, 1995; Shapiro and Varian, 1999). *Technology traps* suggest that many times blind early design decisions later constrain the further expansion of II and become reverse salients (Hughes, 1987). Design for expansion is typically carried out technically and socially in ways that is compatible with the early installed base and its predicted trajectory. Thereby, early design decisions can later block the expansion as new communities join, or technological trajectories change, and create adverse constraints for growth. Such adverse constraints we call *technology traps*. Examples of technology traps are, for example, the need to support archaic computer architectures (e.g. IBM 9010, IBM360 or Intel 8086), operating systems (DOS, Windows95) (Mashey, 2009) or the effects of architecture choices for the growth of the French Minitel system (Cats-Baril and Jelassi, 1994).

Addressing adaptability in II

All systems evolve, but all systems do not adapt equally well. Systems that reach early on lock-in, or exhibit a large number of 'reverse salients' (Hughes, 1987) will fail to do so. According to CAS, highly adaptable systems are characterized by the increased variety achieved through high modularity: 'variation is the raw material for adaptation' (Axelrod and Cohen, 1999: 32). In other words, the larger the variety of agents and pathways for evolution, the more design alternatives can be tried out, and the more agents learn (Holland, 1995; Benbya and McKelvey, 2006). Accordingly, the larger the variety of IT capabilities and the larger the number of II designers the larger is II adaptability. At the same time a certain level of order is necessary in order to maintain stability in the design context. This stability is achieved through modularity. According to CAS, modularity creates a balance between variety and order by localizing the change and permitting fast and deep change in parts of the system. Engineered as well as living systems must thus be modular to remain robust and at the same time to generate variety (Simon, 1969; Baldwin and Clark, 2000; Wagner, 2007). Adaptation becomes optimal at 'the edge of chaos' (Stacey, 1996), where variety generation and modularity are balanced.[7] To wit, traditional application design theory assumes the complete order of stasis as designers are assumed to control all the system states during the design. In contrast, random order excludes the possibility for any design as no order can be detected in the context. Hence, in designing *for* II, designers need to establish a self-organization, where the II will remain 'at the edge of chaos.'

Design rules to manage dynamic complexity – the Internet case

The design principles listed in Table 2 guide II designers to conceive their designs in ways where they can generate 'natural' order at the edge of chaos. To carry out effectively designs that conform to these principles we need to break down the five design principles into *design rules* that govern designer's behaviors influencing specific II components or their environments. In this section we will articulate such design rules. The next section introduces some modularity concepts necessary in stating design rules. The following section offers a summary of their content and reports how we used Internet design to illustrate them. The section after that introduces each design rule with illustrative examples from Internet design.

Modularization of II

In formulating the design rules we need to distinguish properties of IIs that help modularize them. We therefore next define analytical types of (sub) IIs that allow – when composed *together* – the generation of *modular* IIs. We apply recursively de-composition, that is identify separate subsets of IT capabilities within any II, which also are IIs, but which share either a set of common functions and/ or internal or external connections without having strong dependencies with the remaining IIs (Baldwin and Clark, 2000; Kozlowski and Klein, 2000).

We will first split IIs into *vertical application* IIs and of *horizontal support* IIs. The former will deliver functional capabilities, which are deployable directly by one or more user communities. An example of application capability would be e-mail. The latter – support infrastructures – offer generic services often defined in terms of protocols or interfaces necessary in delivering most, if not all application services. They are primarily deployed by designer communities while building application capabilities[8] and include capabilities for data access and identification (addressing), transportation (moving) and presentation (formatting). They also constitute part of the installed base, which II designers need to take into account when bootstrapping an application infrastructure or making changes in support IIs.

We can further recursively decompose both application and support IIs. Thus, any II can be split into its application and support infrastructures until a set of 'atomic' IT capabilities are reached (per recursive definition of II). In addition, any support infrastructure can be split into *transport* and *service* IIs. This split is justified as transport infrastructure is necessary to make any service infrastructure work. The transport IIs offer data or message transportation services like the UDP/TCP/IP protocol stack (Leiner *et al*., 1997). On the other hand service infrastructures support, for example, direct addressing, service identification, service property discovery, access and invocation, or security capabilities. They become useful when IIs start to grow in complexity and scale, and designers need more powerful capabilities to configure application capabilities. A classic example of a service II is the Domain Name Service (DNS) in the Internet, which maps mnemonic identifiers like amazon.com to varying length bit representations, that is IP addresses. Both application and service IIs can be finally linked together horizontally through *gateways*. These offer flexible pathways for II expansion and navigation (Hanseth, 2001; Edwards *et al*., 2007). An example of a gateway would be an IT capability, which supports multiple e-mail services running on different e-mail protocols.

Design rules for dynamic complexity in II's

Derivation and summary of the design rules
In total, we propose 19 design rules for II dynamic complexity shown in Table 3. Overall, the design rules characterize: (1) appropriate ways to organize and relate II components technically and socially (modular design, organize recursively) that address dynamic complexity similar to Baldwin and Clark's (2000) design rules; (2) desirable properties of specifications of II components (e.g. simplicity), (3) desirable sequences for design (e.g. design one-to-many IT capabilities before many-to-many IT capabilities) (4) desirable ways to relate II specifications and associated components to one another (modularity, recursive application).

Illustrating II design rules- the case of Internet
Below we illustrate the deployment of each design rule by referring to episodes of Internet design. We chose Internet design as an illustration as its design history offers rich insights into the situated application of the proposed design theory 'in use.' We chose to illustrate the theory with the design of Internet, because it qualifies as a grand 'success' story about II design *par excellence* (Table 4). By any criterion its design involved cultivating an II, which is shared, open and heterogeneous, organized recursively and operates without centralized control. We also content that the success of Internet testifies to the plausibility of the design rules discussed below.

We gleaned the design rules by content analyzing design episodes, that is, moments when new IT capabilities were added, modified, expanded or purged in the Internet regime. A review of these design situations permit us generalize identified design rules by triangulating data with the emerging theory (Eisenhardt, 1989). Documents like the Internet Engineering Task Force (IETF) rules for Requests For Change, but also the *credo* coined in Clark's famous speech in 1992: 'We reject: kings, presidents, and voting. We believe in rough consensus and running code' (Russell, 2006), and personal biographies all exemplify the behaviors of the Internet designers and consequently their design theories-in-use. To this end we probed Internet standardization archives and primary secondary sources on Internet history – especially Abbate's (1999) excellent narrative. We also examined personal accounts of Internet design (Leiner *et al*., 1997; Berners-Lee and Fischetti, 1999), and recent scholarly analyses of the Internet growth (Tuomi, 2002). Finally, we interviewed Robert Kahn, one of the original developers and sponsors of Internet protocols. As a result we were able to solicit 19 design rules as summarized in Table 5 founded on CAS that had 'worked' during Internet design.

For sure, not all designers at all times and consciously followed these rules. But many of them acted consistent with these rules. For example, they underpin many common 'interaction rules' in the Internet design ecology. Many key figures have also openly embraced proposed design principles. For instance, throughout Internet's history, from Kahn's and Cerf's initial design to the creation of BitTorrent, the Internet designers have favored bottom-up, experimental design and utilization of network effects. At the core was also the recognition of high uncertainty related to desired IT capabilities. Bob Kahn noted vividly this:

> They (DoD) didn't have a problem. And that's why it's so hard for those kinds of things to actually get in motion. If you're saying, 'Can I imagine a problem that somebody might have at some unspecified point in the future?' Absolutely, that was what was driving it. And, so you had to really trust what was in your mind's eye. And that was the basis on which the internal justifications were eventually made. But it took a while to get there. (Kahn, 2006)

Therefore, he argued that the only way to design was experimental:

> But when you're dealing with something as really state of the art, it's hard to know what to build upfront because a lot of what makes it what it is a function of, you know, iterating with users and feedback and allowing the system to evolve and grow and to see how it would work. This is not something that most people, who are, you know, in management chain, are very uncomfortable with because they don't exactly know what to expect. So it's very hard for those people, you know, to deal with those kinds of creative processes. (Kahn, 2006)

Design rules for dynamic complexity in the design for II

We next discuss in detail how the 19 design rules in Table 3 were inferred from the CAS theory offering a 'because of' justification for the design principles. To wit, each design rules offers a falsifiable statement of design outcomes to validate the theory. By analyzing whether the designer followed the rule and related design outcomes, we can determine whether the rule following did not lead to the predicted outcome thus falsifying (partly) the proposed design theory.

Design rules for the bootstrap problem
Often new IT capabilities are not adopted despite their novelty, because users wait others to adopt first: early adopters face high risks and costs, but few benefits. In light of CAS, an II designer must generate attractors to propel users to adopt the IT capability so that its growth will reach a momentum (Hanseth and Aanestad, 2003). We observe three design principles decomposed into 12 design rules that help generate and manage such attractors (see Tables 3 and 5).

Design rules for principle #1: Design initially for direct usefulness: Early users cannot be attracted to IT capabilities reasons like the size of their installed base. Therefore, we need design rules that foster relationships between the proposed IT capability and user adoption. Therefore, a small user population needs to be identified and targeted (Design Rule 1 (DR1[9])). The proposed IT capability has to offer the group *immediate* and *direct* benefits (DR2). Because first adopters accrue high adoption costs and confront high risks, the IT capability to-be-adopted must be

Table 3 Design rules for dynamic complexity in the design for IIs

Design problem	*Element of CAS*	*Design principles*	*Design rules*
Bootstrap problem Design goal: Generate attractors that bootstrap the installed base	Create an IT capability that can become an attractor for the system growth.	1. Design initially for direct usefulness.	DR1. Target IT capability to a small group DR2. Make IT capability directly useful without the installed base DR3. Make the IT capability simple to use and implement DR4. Design for one-to-many IT capabilities in contrast to all-to-all capabilities.
	Avoid dependency on other II components that deflect away from the existing attractors Use installed base as to build additional attractors by increasing positive network externalities	2. Build upon existing installed bases	DR5. Design first IT capabilities in ways that do not require designing and implementing new support infrastructures DR6. Deploy existing transport infrastructures DR7. Build gateways to existing service and application infrastructures DR8. Use bandwagons associated with other IIs
	Exclude alternative attractors by persuasive tactics Offer additional positive network externalities by expanding learning in the user community	3. Expand installed base by persuasive tactics to gain momentum	DR9. 'Users before functionality' – grow the user base always before adding new functionality DR10. Enhance any IT capability within the II only when needed DR11. Build and align incentives so that users have real motivation to use the IT capabilities within the II in new ways DR12. Develop support communities and flexible governance strategies for feedback and learning
Adaptability problem Design Goal: Make the system maximally adaptive and variety generating as to avoid technology traps	Build capabilities that enable growth based on experience and learning Use abstraction and gateways to separate II components by making them loosely coupled	4. Make the IT capability as simple as possible	DR13. Make the II as simple as possible in terms of its technical and social complexity by reducing connections and governance cost DR14. Promote partly overlapping IT capabilities instead of all-inclusive ones.
	Design IT capabilities and their combinations in ways that allow II growth Use evolutionary strategies in the evolution of II that allow independent incremental change in separate components Draw upon II designs that enable maximal variations at different components of the II	5. Modularize the II	DR15. Divide II recursively always into transportation, support and application infrastructures while designing the II DR16. Use gateways between standard versions DR17. Use gateways between layers DR18. Build gateways between infrastructures DR19. Develop transition strategies in parallel with gateways

Table 4 Internet as an II

	Internet as an II
Structural properties	
Organizing principles	Internet is composed of multiple layers of distinct IT capabilities that carry out similar functions at different layers (e.g. transport and application layer). It consists of or draws upon multiple platforms, IT capabilities and social groups that design, implement and maintain its functionality.
Control	The control of Internet design is distributed among a large set of designers, user communities and forms of governance. The control of different capabilities is separated and distributed and the control forms are loosely coupled through architectural principles. Control forms vary among different communities (IETF, W3C or OASIS) as well as governance structures (Nickerson and Zur Muehlen, 2006; Russell, 2006).
Emergent properties	
Shared	Shared by an increasingly growing number of heterogeneous user communities, designers, regulators and other social actors.
Open	Any new IT capability, designer or user group can be added as long as it conforms to the architectural principles of Internet and thus abstracts data transfer into a transfer of data streams to a specified set of IP addresses.
Heterogeneous	Internet has grown immensely in heterogeneity both socially and technically since its inception and during its exponential growth.
Evolving	Evolving set communication and distributed computing capabilities For any set of users at any time Internet is seen as a distinct set of capabilities (telnet, ftp, smtp, http, etc.) that are available. Internet evolves because this set has grown significantly during its evolution integrating new users and design communities.

simple, cheap and easy to learn (DR3). Here cheap is defined in relation to both design and learning costs. Simple means that the design covers only the essential functionality expected and the capability is designed so that it is easy to integrate the IT capability with the installed base. Significant user investments cannot be expected because a small user base does not contribute either to the demand or the supply side economies of scale.

IT capabilities have varying impacts on the scale of increasing returns and the amount of positive feedback. They vary significantly between capabilities where every user interacts symmetrically with every user (like e-mail) and capabilities where one user interacts uni-directionally with the rest. Capabilities can also have multiple possible implementation sequences. In general, IT capabilities supporting asymmetrical interactions (one-to-many) and thus less dependent on network effects should be implemented first as the growth can be promoted locally (DR4). These capabilities have lower adoption barriers as they do not need to reach a critical mass to generate fast adoption.

The Internet's success has been widely attributed to its successful bottom-up bootstrapping (Leiner *et al*., 1997; Abbate, 1999; Tuomi, 2002; Kahn, 2006). Though early on Internet designers built bold scenarios of how the future of telecommunications would unfold (Tuomi, 2002), the early uses of packet switching were targeted to small groups of researchers, who were interested in accessing powerful and expensive computers (DR1). The aim was to provide a limited range of directly useful IT capabilities: remote login and file transfer (DR1). Among these capabilities, remote login was a perfect choice, because each user could adopt it independently from others and users had the skills and motivation to do so (DR3, DR4). While the number of users grew, they could start share data through file transfer. Later on, new capabilities have been introduced in the same way (DR2). E-mail, for instance, was originally developed to support communications between persons responsible for maintaining the network when only four computers were connected to it (Abbate, 1999) (DR1 and DR2). The design of transportation services (TCP) followed also an evolutionary approach as multiple versions of increasingly complete protocols for TCP and IP were implemented in the early 1980s (DR3).

Design rules for principle #2: build on installed bases: The second principle promotes connections with the existing installed base during design time. The II designer should thus design toward existing support infrastructures that the targeted user groups use (DR5). If an IT capability is designed so that it requires a new support infrastructure, this will erect heightened adoption barriers as, per our definition, the support infrastructure will be *sui generis* an II, which then needs to be bootstrapped with high learning barriers (Attewell, 1992). As noted, transport infrastructures form the base for implementing II while the need for service infrastructures depends on the size and sophistication of application capabilities, or the size of installed base. While the installed base remains small, the II does not need advanced service infrastructures. The II designer

Table 5 Design principles and rules for Internet

Challenge: Bootstrap problem

Design principle	*Design rules*	*Followed*	*Evidence from Internet design history*
1. Design initially for usefulness	DR1. Target IT capability to a small group	✓	Designed originally as a support capability for Defense Advanced Research Projects Agency (DARPA) researchers to share expensive and centralized computing resources
	DR2. Make IT capability directly useful without an installed base	✓	Designed as application for log in and file downloads The developed IT capability was well suited for experimenting with distributed Unix and LAN technologies, in which originally reaching large installed base was not an issue
	DR3. Make the IT capability simple to use and implement	✓	Obtain experience based on the use of simple prototypes and capabilities. This has been an important design principle in the Internet community, in particular during the early adoption (Leiner *et al.*, 1997; Abbate, 1999)
	DR4. Design for one-to-many IT capabilities in contrast to all-to-all	✓	Remote login as a first capability
2. Build upon existing installed bases	DR5. Design IT capability that does not depend on new support infrastructure	✓	A principal design rule in implementing the TCP/IP protocol stack was to make it run on top of multiple underlying physical access layers: telephone, radio, satellite, LAN technologies, etc. (Abbate, 1999)
	DR6. Deploy existing transport infrastructures	✓	All early capabilities were introduced without any new related transport infrastructures
	DR7. Build gateways to existing service and application infrastructures	✓	Gateways were developed for example to other e-mail protocols and other IT capabilities. Recently gateways (e.g. CGI) to databases and application capabilities have been important in making Web-services useful
	DR8. Use bandwagons associated with other IIs	✓	During the 1980s, Internet increased its acceptance with the diffusion of workstation/Unix/LAN technologies
3. Expand installed base by persuasive tactics to gain momentum	DR9. Users before functionality	✓	Many Internet capabilities have been added when their needs have been recognized. The development of TCP/IP and its new versions IPv6, the introduction of DNS, enhancement of the web by XML and CCS are examples of this
	DR10. Enhance the IT capability within the II only when needed	✓	Internet offered low cost and innovative ways for many scientific and engineering communities to communicate and share information and build associated services. Important allies were research funding agencies and research communities
	DR11. Build and align incentives as needed	✓	Development and use intertwined to build communities. A large community (critical mass) was, e.g., built originally to learn from distributed computing. The same applies to Web, Instant messaging or multicasting
	DR12. Develop support communities	✓	Developers and users are both innovators for IT capabilities and organized support communities to do so

Table 5 Continued

Challenge: Bootstrap problem

Design principle	*Design rules*	*Followed*	*Evidence from Internet design history*
4. Make the design of IT capability as simple as possible	DR13. Make the II in terms of its technical and social complexity as simple as possible	✓	This rule was explicitly stated early on (RFC, 1994) and has been important throughout the design history of the Internet
	DR14. Promote partly overlapping IT capabilities instead of all-inclusive ones	✓	This rule is enabled by the principle that standards are open and any one can design at the edge new functionality resulting in multiple overlapping capabilities
5. Modularize the II	DR15. Divide infrastructure recursively into transportation, support and application infrastructures	✓	The architectural principle of 'end-to-end' architecture that promotes independence and modularization (David, 2001) Internet is composed of a large number of separate capabilities, sub-infrastructures that are established and operated by independent actors including ISPs, etc.
	DR16. Use gateways between specification versions	✓	The paradigmatic example of this is the use of tunneling in the transition from version 4 to 6 of the IP protocol
	DR17. Use gateways between layers	✓	This is key architectural principle of Internet that separates applications, transport, addressing and physical connections. All these layers are connected through open gateways
	DR18. Build gateways between infrastructures	✓	This was originally used to connect different networks (BITNET, Decnet) with Internet e-mail protocols. The same took place with AOL and Progidy
	DR19. Develop transition strategies in parallel with gateways	✓	Consideration of transition strategies is carefully introduced into the new versions of the protocol specifications

should therefore design toward the simplest possible service infrastructure (DR6). Next, capabilities associated with separate service and application infrastructures should be connected, when possible, through gateways increasing connections between isolated user communities and benefitting adopters with larger positive network effects (DR7). As the designers link the new IT capabilities to the existing IIs, they need to take into account the speed and direction of the adoption of IT capabilities in neighboring infrastructures, and capitalize on their bandwagon effects (DR8).

The Internet's early success resulted from exploiting established infrastructures as transport infrastructures (DR5) when TCP/IP was first implemented using modems over the telephone lines (Abbate, 1999). In addition, each adopted capability has served to develop more advanced capabilities (Abbate, 1999) (DR6). Currently, the Internet provides, for example, capabilities for electronic commerce including transaction support (e.g. EbXML[10]), identification support (e.g. digital certificates) or security (e.g. SET) built as separate capabilities on top of TCP/IP and http (Faraj *et al*., 2004; Nickerson and Zur Muehlen, 2006). Another example is the initial growth of Internet's service infrastructures (DR6). In the beginning there was none and their need was discovered later when new service capabilities started to grow. Yet, the scale of Internet was still relatively small so that it was easy to design DNS capabilities and link it to a (now) stable transportation infrastructure. Later on DNS became critical as it increased flexibility of use through the management of dynamic IP addresses (DHCP). The Internet designers have also increased the installed base through gateways (DR7). The expansion of the Web functionality is a case in point. The Web was originally thought to be useful for static information provisioning so that HTML tagged files could be downloaded using the http protocol (Tuomi, 2002). A significant added value for Web was created by building gateways that leveraged upon data residing in organizational databases. This added dynamic or 'deep' web features: a call to data base could be now embedded in HTML as defined by Common Gateway Interface (CGI) specifications,[11] and later expanded with Java standards (RMI[12]).

Design rules for principle #3: expand installed base with persuasive enrollment tactics: After establishing the first attractor (usefulness), the II designers have to sustain growth. Therefore, when a simple version of the IT capability is available, the II designer needs to seek as many users as possible (DR9). This principle is captured well in a slogan: 'users before functionality' emphasizing the criticality of generating positive network effects: the IT capability derives its value from the size of its user base – *not* from its superior functionality. New functionality should be added only when it is truly needed, and the original capability obtains new adoption levels so that the proposed capability will have enough users willing to cover the extra cost of design and learning (DR10). Many times useful new functionality emerges when users start deploy the IT capability in unexpected ways through learning by doing and trying, or re-organizing the connections between the user communities and the IT capability (DR11). A growing installed base urges II designers to find means to align heterogeneous user interests and persuade them to continue to participate in the II. One approach is to use the installed base as a source of useful learning by creating user communities that offer feedback. This helps introduce new capabilities based on feedback and unexpected actor interactions (DR12) (Tuomi, 2002; Zimmerman, 2007).

Many capabilities during Internet design were established at times when the capabilities could be expected to work satisfactorily and serve a useful purpose (DR9). As a result increasingly sophisticated application capabilities emerged including Gopher (Minnesota), WAIS (Cambridge, Mass) and irc (University of Oulu) (Rheingold, 1993). As a result Internet has grown over the years enormously in terms of new services and protocols. Typically these capabilities emerged as local community responses to an identified local need (DR10), and only a tiny fraction of the Internet's current protocol stack was part of the initial specifications (DR10). Main reason for this was that most innovations took place at the 'edge' as design capability and application functionality were early on moved to the network boundary. New capabilities could be conceived and tried out whenever a user with a 'problem' and enough transportation capability could leverage upon the new functionality (Tuomi, 2002). Internet was also widely adopted by computer science and associated engineering communities as their research-computing infrastructure (DR11 and DR12). The open packet switching standards turned out to be perfectly suited for the research vision shared by this movement (Kahn, 2006).

Design rules for the adaptation problem

When the bandwagon starts rolling, the II designers need to guarantee that the II will grow *adaptively* and re-organize constantly with new connections between II components. *Ad hoc* designs, which were originally created for early users will now threaten to create technology traps. If designers continue to generate highly interdependent and local IT capabilities, the whole system will become inflexible and reach a stasis. In contrast, if IT capabilities are organized modularly through loosely coupled 'layers,' which can change independently, this will generate higher component variation for successful adaptation. The following two design principles decomposed into seven rules offer guidance to promote modularity.

Design rules for principle #4: make the organization of IT capabilities simple: The first principle asks for the use of simple architectural principles during the initial design of the IT capabilities (DR13). It is easier to change something that is simple than something that is complex. What makes a collection of IT capabilities simple or complex is a function of its technical complexity as defined by the number of its technical elements, their connections and rate of change (Edwards *et al*., 2007). Therefore following information hiding, simple interface protocols and functional abstraction can help make the design simple. But, just as important is it to recognize the socio-technical complexity of the design space: the number and type of connections between technical and the social elements. In the lingo of Actor Network Theory (Latour, 1999) the actor network constituted by the II, that is its data elements, use

practices, specifications and their discovery and enforcement practices, the relationships to other infrastructures, the multiplicity of developers, the role of organizations, the variety of users, the regulatory bodies etc. – and a myriad of links between all affect what can be changed and how (Star and Ruhleder, 1996; Latour, 1999). Simpler actor networks can be created by making them initially as small as possible, and keeping them loosely connected, and avoiding confrontations with competing networks. This is achieved by pursuing separate specifications for distinct domains and separating the concerns of different social and technical actors through functional abstraction (Tilson, 2008). Limiting the functional scope of application infrastructures to a minimum keeps the related infrastructures separate. Decomposing service IIs into a set of layers and separating their governance achieves the same goal. These principles decrease the technical complexity of specifications but, more importantly, reduce their social complexity. Finally, designs should promote partly overlapping IT capabilities instead of all-inclusive ones. This increases variance and stimulates innovation at different pockets by making it operate at 'the edge of chaos' (DR14).

The principles of early Internet design promoted simplicity (DR13). Its protocols were lean and simple, and therefore had less ambiguity and errors. As a result the implementations were simpler, and easier to test and change. Origins of this approach date back to early designs, which confronted early on the challenge of how to promote change, but at the same time to avoid technology traps. This was expressed early in the Internet's specification approach:

> From its conception, the Internet has been, and is expected to remain, an evolving system whose participants regularly factor new requirements and technology into its design and implementation. (RFC, 1994: 6)

This vision was opposite to traditional design strategies in the telecommunication industry followed in the design of the ISO/OSI protocol stack where designers assumed one homogeneous, complete and controllable network, which had to be completely specified (Abbate, 1999; Russell, 2006). This difference was later at the center of the controversy between the Internet community and the ISO/OSI committee (Schmidt and Werle, 1998; Russell, 2006). The OSI standardizers argued that Internet lacked critical functions; in contrast, the Internet community advocated technical simplicity and pragmatic value. As Kahn observed:

> So the only way that you could ever get anything to be a standard was: you had to have built it first; it had to be deployed; and basically, people would speak by adoption. So the things that became standard … were the things that were starting to become in widespread use and they would eventually become standards when they were already used. This is the equivalent of ratification after the fact, the standard is simply a means of ratifying what has become in widespread use … a very different approach than specifying upfront and hoping people will build it. (Kahn, 2006)

Many scholars have attributed the demise of the OSI to its disregard to this pragmatic approach (Rose, 1992; Stefferud, 1994). Finally, Internet always promoted designs that were partly overlapping increasing variety. For example it has generated several transportation protocols, e-mail protocols, information distribution protocols and so on (DR14).

Design rules for principle #5: modularize the II: As noted, II designers need to organize modularly capabilities into loosely coupled sub-infrastructures (Parnas, 1972; Baldwin and Clark, 2000). Therefore, IIs should be decomposed recursively into separate application, transport and service sub-infrastructures (DR15). Each II interface must hide mechanisms that implement these capabilities as to maintain loose couplings between the connected IIs. IIs need to be also decomposed vertically into independent neighboring application infrastructures, and II designers need to build gateways to connect them. Consequently, gateways must connect regions of II that run different versions of the same IT capabilities (DR16), or between different IT capability layers, for example, transport or service (DR17), or between several dedicated application infrastructures (DR18) (Edwards *et al*., 2007). Finally, transitions between incompatible IT capabilities need to be supported by navigation strategies that allow local changes in different versions of the IT capability that run on the current installed base (DR19).

One reason for the speed of innovation in Internet was its initial modular design (DR15) (Tuomi, 2002). The Internet's simple end-to-end architecture, which puts the 'intelligence' into the end nodes, has proven to be a critical for its adaptive growth (DR15, DR16, DR17) (Abbate, 1999; David, 2001). The design stimulated continued local application or service infrastructure innovation laid on top of separate transportation infrastructure of TCP/IP or UDP (DR15) (Rheingold, 1993; Tuomi, 2002). Each of these capabilities was designed independently and its design decisions were insulated from potential changes in the underlying transport infrastructures. They were also governed separately.[13] The erection of the W3C and governance of the web service community forms a case in point (Berners-Lee and Fischetti, 1999). Gateways continue to play a critical role in the evolution of Internet and extensive use of gateways has prevented designers to act like 'blind giants,' and made early decisions easier to reverse (Hanseth, 2001). Multiple gateways prevail, for instance, between the Internet's e-mail service and proprietary e-mail protocols (DR17). Another important family of gateways has been built between the Internet's access services and organization's applications and databases through web servers (DR18). Over the years Internet protocols have been revised and extended (DR16, DR19). One example is the revision of the transportation protocol from IPv4 to IPv6. The need to add new capabilities while attempting to overcome installed base inertia has been a major design challenge (RFC, 1994; Monteiro, 1998; Hovav and Schuff, 2005). Between 1974 and 1978, four versions of the IP protocol were developed in fast experimental cycles until IPv4 was released (Kahn, 1994). For the next 15 years IPv4 remained stable. In the early 1990s Internet's address space was expected to run out due to the Internet's exponential growth. Moreover, the addressing scheme in

IPv4 did not support multicasting and mobility. This triggered a new round of designs to deliver a new IP version called IP version 6.[14] The final version, however, fulfilled only few of the original requirements – the most important one being the extension of the reverse salient – address space – to awesome 2^{128} addresses.[15] The most important criterion in accepting the final specifications was in determining mechanisms that would introduce the new version in a stepwise manner (DR19), though initially this was not at all in the requirements (RFC, 1995; Steinberg, 1995; Hovav and Schuff, 2005).[16]

Concluding remarks

Today's IT systems involve complexity that extends beyond what can be addressed by traditional design approaches. Accordingly, we need to theorize in fresh ways how to design complex IT systems. To this end we have formulated a design theory based on CAS theory that tackles IIs' dynamic complexity. The theory was derived by scrutinizing design histories of large infrastructures, a review of CAS theory principles and illustrated by the analysis of Internet exegesis. The theory formulation follows an approach similar to Lindgren *et al.* (2004), and Markus *et al.* (2002). By formulating this design theory we make a contribution to IS Design and Software Engineering research on how to develop large and complex ICT solutions viewed as IIs. We do so by drawing extensively on prior research on II evolution and soliciting the empirical insights into a coherent design theory.

The proposed theory defines its unit of analysis, its essential properties and related kernel theory – CAS – as to derive five design principles, and19 design rules. It recognizes and draws upon earlier research on IIs (Kling, 1992; Star and Ruhleder, 1996) by observing pivotal relationships between technical and social elements, and their dynamic interactions. In contrast to earlier II research (Freeman, 2007; Zimmerman, 2007), the proposed theory adopts the viewpoint of designers: how to 'cultivate' an installed base and promote its dynamic growth by proposing *design rules* for II bootstrapping and adaptive growth. Opposed to other design theories and methodologies, which are all 'design from scratch' approaches, our design theory puts the installed base at the center: II development is about how to create a self-reinforcing installed base by drawing upon existing ones, and how to avoid being trapped by the force of the installed base.

All theories are incomplete (Weick, 1989) and so is our proposed theory. Hence, the key question is not to ask whether the proposed theory is incomplete (as it will be), but rather: what are the implications of its limits? We will address this in two ways: (1) how to address incompleteness in the scope of our theoretical formulation, (2) and how to improve consequently its external validity. We drew upon CAS as to account for the feedback-based growth within complex socio-technical systems. The principles that underlie CAS are widely accepted as illustrating how complex systems evolve. Our contribution has been to revise CAS into a form that can be utilized in design thinking in the context of IIs. In doing so we drew upon extant II and other literatures to propose a set of falsifiable design rules (Baldwin and Clark, 2000). Unfortunately, our theory refinement still does not offer detailed recommendations of how to decide in specific contingencies about II designs. We are confident that in some situations the theory has limited applicability. For example, the design of IIs that necessitate single 'point' coordination through significant early investments like the design of wireless systems may follow a more centralized specification driven approach. Likewise, the theory does not help estimate the economic consequences of choosing between infrastructural alternatives (Fichman, 2004). The theory is also limited in its scope. It says nothing about the politics during II design and how a designer can cope with the power. To do so, we would have to integrate the theory with theories that recognize power like actor network theory (Latour, 1999), or institutional theory (Scott, 2001). Finally, it cannot account for all critical features of II design like security.

The proposed theory was kept simple as we preferred generality over accuracy (Weick, 1989). Consequently, it is composed of a small set of concepts offering design abstractions across a set of IT capabilities and their growth patterns. These concepts hide differences by applying abstraction and composition (Kozlowski and Klein, 2000). Therefore, the simplicity of the theory comes at a cost: its design rules offer no silver bullet for prediction, and it has at most a pragmatic legitimacy (Robey, 2003). Its knowledge claims can thus improve II designs instead of suggesting the 'optimal' design. One use of the theory is in explaining *post hoc* to what extent design processes with observed outcomes followed or did not follow principles derived from the CAS theory. Another use is to guide designs through enacting rules that promote increased II adaptability by stating what 'thou shall not,' that is: (1) what *not* to *assume* (e.g. complete control), or (2) what *not* to *do* (keep it simple stupid!).

We illustrated the theory through an investigation of successful design episodes around Internet. In this application we viewed the theory through its use *utility* – that is did the enactment of the design rules lead to stated design goals? These principles and rules have also been applied during the successful design of national IIs for health care in developing countries including South Africa, Ethiopia, Tanzania, Nigeria, India and Vietnam (Braa *et al.*, 2007). These programs emphasized bottom-up and iterative development and relied on simple solutions using flexible standards. Our question for all these experiences is: did the theory make a difference and how would we evaluate it under counter-factual conditions? Had it made a difference in the Internet case had the designers *not* pursued the design rules?[17] Naturally, we can never be completely certain about this as we cannot carry out a new 'experiment' under the same conditions. We content, however, that, had designers followed alternative design rules, the Internet would not have been bootstrapped as effectively. Many other II designs with similar goals followed different rules, but failed despite huge institutional backing and deep resource commitments (like ISO/OSI). We have neither examined situation where *not following* the design rules led to successful outcomes, or where following the design rules led to failures.[18]

In future, we will expand the proposed theory by analyzing other II design episodes. Some candidates

are: the digital transformation in industries including architecture and construction (Boland *et al*., 2006), health care (Hanseth and Monteiro, 1997) or financial services (Markus *et al*., 2006). Another route is the creation of service infrastructures for web services (Nickerson and zur Muehlen, 2006), and broadband mobile services (Yoo *et al*., 2005; Tilson and Lyytinen, 2006).

Our theory has significant practical implications. If its design rules were widely adopted, the II designers would have to prefer continuous, local innovation, to increase chaos, and to apply simple designs and crude abstractions. This change is not likely, as design communities are often locked into institutional patterns that reinforce design styles assuming vertical control and complete specifications. The best example of this is perhaps Tim Berners-Lee's legacy. He successfully introduced the Web by following design rules that address dynamic complexity, but later changed into a specification-driven approach during the development of the semantic web. This process, however, has been more onerous. The lesson learned is: designers learn often superstitiously (March, 1991). We hope, however, that this essay highlights why changing such superstitions makes a lot of sense in today's design.

Acknowledgements

The paper has benefited from the helpful comments of Nick Berente, Sean Hansen, David Tilson, Youngjin Yoo, Vallabh Sambamurthy, Bo Dahlbom, Eric Monteiro, Margunn Aanestad, Petter Nielsen, Lynne Markus and Omar El-Sawy, the senior editor Jannis Kallinikos and two reviewers for their constructive comments. We also thank faculties at the Helsinki School of Economics, Umea University, University of Oslo and Case Western Reserve University for constructive feedback. Finally, we want to thank all the people who participated in the study for sharing their experience and thoughts.

Notes

1 In one sense infrastructures just evolve, if we rely on biological metaphor and the idea of 'blind' mutation. But, because infrastructures are artifacts created by intentional action, we prefer to use the term 'design for' instead of 'design of' as designer's behaviors matter how, and to what extent the infrastructure can evolve. We have elsewhere proposed the term 'cultivate' for this type of design activity.

2 We use the term information infrastructure as a symmetrical concept to that of an application, that is an information system. Both are socio-technical artifacts, and thus 'designed.' Both consist of elements of hardware, software and data that are integrated into a suite of IT capabilities and designed, used and regulated by social groups. But their behavior, design parameters and characteristics that define good 'designs' are different as argued below.

3 Consider, for example, the design of the aviation application embedded in a modern airplane like Airbus A380. Its specifications are derived from a host of avionics engineers, regulators, airline managers and so on, and developed and controlled by a group of developers who are organized into a hierarchy. The Airbus 380 software is therefore also useful immediately. It will, however, gradually obtain new features that differentiate it from its initial specification. For example, the navigation systems of A380 may have to communicate with new air-traffic control systems and be integrated with new media and communication software, or airplane maintenance and control systems. During its use Airbus380 avionics software will thus evolve in unanticipated ways based on user learning, regulatory demands and innovation around IT that support new avionic tasks. Accordingly, the Airbus A380 applications, when used over time, become connected with multiple external IT capabilities that expand in unanticipated ways. As a result the applications become a critical component in a complex web of interlocked set of IT capabilities in the modern avionics acquiring infrastructural features.

4 This is an idealized view as many times user needs are unknowable, poorly expressed or change too fast to truly address them.

5 II specifications are often called standards and regarded essential in building IIs (Star and Ruhleder, 1996; Edwards *et al*., 2007). Standards are shared and agreed upon specifications among a set of communities. We deem them not analytically necessary for II design. They are, however, one of the most effective means to coordinate the distributed design of IIs, and they play a prominent role to expand, coordinate and deploy IT capabilities in a distributed manner.

6 The theory has also been influenced by our experiences in designing other types of IIs. These include mobile infrastructures, ERP implementations in large organizations and electronic patient record infrastructures.

7 For example, Google follows 70-20-10 rule to maintain a balance between order and chaos. They use 70% of their resources and attention to improve the current order of the core businesses, 20% to work on related and incremental adaptations and 10% in other, non-related new 'permutations.'

8 This principle is similar to decomposition of dedicated IT applications if we distinguish between user defined computational functions (e.g. computing a salary), and generic horizontal system functions (e.g. retrieving or storing an employee record).

9 This refers to the rule label in Table 3.

10 See http://www.ebxml.org/.

11 See http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

12 See http://java.sun.com/products/jdk/rmi/.

13 This was not always done without friction (see, e.g., Nickerson and Zur Muehlen, 2006).

14 For a definition see http://playground.sun.com/pub/ipng/html/ipng-main.html.

15 It has been later observed that other 'workarounds' like DHCP and NAT actually could circumvent the address space problem and the value of IPv6 in this solving the original requirement has been questionable.

16 See, e.g., http://www.ipv6.org/.

17 Falsification principle followed is contingent in the sense that we can never be sure that the strategy would work successfully in all future cases, that is, the theory remains always falsifiable.

18 We have no such evidence after analyzing multiple cases.

References

Aanestad, M. and Hanseth, O. (2002). Growing Networks: Detours, stunts and spillovers, Proceedings of COOP 2002, Fifth International Conference on the Design of Cooperative Systems, St. Raphael, France, 4–6th June 2002.

Abbate, J. (1999). *Inventing the Internet*, Cambridge, MA: MIT Press.

Agresti, W.W. (1986). The Conventional Software Life-Cycle Model: Its evolution and assumptions, in W.W. Agresti (ed.) *New Paradigms for Software Development*, Washington, D.C: IEEE Computer Society.

Arthur, W.B. (1994). *Increasing Returns and Path Dependence in the Economy*, Ann Arbor: The University of Michigan Press.

Attewell, P. (1992). Technology Diffusion and Organizational Learning: The case of business computing, *Organization Science* **3**(1): 1–19.

Axelrod, R.M. and Cohen, M.D. (1999). *Harnessing Complexity: Organizational implications of a scientific frontier*, New York: Free Press.

Baldwin, C. and Clark, K. (2000). *Design Rules*, Cambridge, MA: MIT Press.

BCS/RAE (2004). The challenges of complex IT projects, British Computer Society and Royal Academy Engineering Project [www document], http://www.bcs.org/upload/pdf/complexity.pdf (accessed August 2009).

Benbya, H. and McKelvey, B. (2006). Toward Complexity Theory of Information System Development, *Information, Technology and People* **19**(1): 12–34.

Benkler, Y. (2006). *The Wealth of Networks. How Social Production Transforms Markets and Freedom*, New Haven, CT; London: Yale University Press.

Berners-Lee, T. and Fischetti, M. (1999). *Weaving the Web-The Original Design and Ultimate Destiny of World Wide Web by Its Inventor*, San Francisco: Harper-Collins.

Boehm, B.W. (1976). Software Engineering, *IEEE Transactions on Computers* **C-25**(12): 1226–1241.

Boland, R., Lyytinen, K. and Yoo, Y. (2006). Path Creation with Digital 3D Representations: Networks of innovation in architecture, engineering and construction, *Organization Science* **18**: 631–647.

Bowker, G. and Star, S.L. (1999). *Sorting Things Out. Classification and Its Consequences*, Cambridge, MA: MIT Press.

Braa, J., Hanseth, O., Mohammed, W., Heywood, A. and Shaw, V. (2007). Developing Health Information Systems in Developing Countries – The flexible standards strategy, *MIS Quarterly* **31**(2): 381–402.

Carpa, F. (1996). *The Web of Life. A new scientific understanding of living systems*, London: Harper Collins.

Cats-Baril, W. and Jelassi, T. (1994). The French Videotext System Minitel: A successful implementation of national information technology infrastructure, *MIS Quarterly* **18**(1): 1–20.

Chung, S., Kelly Rainer, R. and Lewis, B. (2003). The Impact of Information Technology Infrastructure Flexibility on Strategic Alignment and Applications Implementation, *Communications of the Association of Information Systems* **11**: 191–206.

Ciborra, C., Braa, K., Cordella, A., Dahlbom, B., Failla, A., Hanseth, O., Hepsø, V., Ljungberg, J., Monteiro, E. and Simon, K. (2000). *From Control to Drift. The Dynamics of Corporate Information Infrastructures*, Oxford: Oxford University Press.

Contini, F. and Lanzara, G.F. (eds.) (2009). *ICT and Innovation in the Public Sector*, Basingstoke, England: Palgrave Macmillan.

Cordella, A. (2004). Standardization in Action, Paper presented at European Conference on Information Systems, Turku, Finland, 14–16th June 2004.

Damsgaard, J. and Lyytinen, K. (2001). Building Electronic Trading Infrastructure: A private or public responsibility, *Journal of Organizational Computing and Electronic Commerce* **11**(2): 131–151.

David, P.A. (1986). Understanding the Economics of QWERTY, in W.N. Parker (ed.) *Economic History and the Modern Economist*, Oxford and New York: Basil Blackwell.

David, P.A. (2001). The Beginnings and Prospective Ending of 'End-to-End' – An evolutionary perspective on internet architecture, Working Papers 01012, Stanford University, Department of Economics [www document] http://ideas.repec.org/p/wop/stanec/01012.html.

DeMarco, T. (1978). *Structured Analysis and System Specification*, New York, NY: Yourdon Press.

Dooley, K. (1996). Complex Adaptive Systems: A nominal definition, [www document] http://www.public.asu.edu/~kdooley/papers/casdef.PDF (accessed 13th August 2009).

Edwards, P., Jackson, S., Bowker, G. and Knobel, C. (2007). Report of a Workshop on 'History and Theory of Infrastructures: Lessons for new scientific infrastructures', University of Michigan, School of Information [www document] http://www.si.umich.edu/InfrastructureWorkshop/documents/UnderstandingInfrastructure2007.pdf (accessed 15th March 2007).

Egyedi, T.M. (2002). Standards Enhance System Flexibility? Mapping Compatibility Strategies Onto Flexibility Objectives [www document] http://www.tudelft.nl/live/binaries/0b330c26-def4-45e3-a367-43b61bf0ae45/doc/mapping.pdf.

Eisenhardt, K. (1989). Building Theories from Case Study Research, *Academy Management Review* **14**(4): 532–550.

Evans, D.S., Hagiu, A. and Schmalensee, R. (2006). *Invisible Engines: How software platforms drive innovation and transform industries*, Cambridge, MA: MIT Press.

Faraj, S., Kwon, D. and Watts, S. (2004). Contested Artifact: Technology sensemaking, actor networks, and the shaping of the web browser, *Information Technology & People* **17**(2): 186–209.

Fichman, R. (2004). Real Options and IT Platform Adoption: Implications for theory and practice, *Information Systems Research* **15**(2): 132–154.

Freeman, P.A. (2007). Is 'Designing' Cyberinfrastructure – or, even, defining it – possible? *First Monday* **12**(6), June [www document] http://firstmonday.org/issues/issue12_6/freeman/index.html (accessed 8th August 2007).

Funk, J.L. (2002). *Global Competition Between and within Standards: The case of mobile phones*, New York: Palgrave.

Giddens, A. (1984). *The Constitution of Society*, London: Polity Press.

Greenhalgh, T., Stramer, K., Bratan, T., Byrne, E., Mohammad, Y. and Russell, J. (2008). Introduction of Shared Electronic Records: Multi-site case study using diffusion of innovation theory, *British Medical Journal*, (Clinical research edn) **337**: a1786.

Gregor, S. (2006). The Nature of Theory in Information Systems, *MIS Quarterly* **30**(3): 611–642.

Grindley, P. (1995). *Standards, Strategy, and Politics. Cases and Stories*, New York: Oxford University Press.

Grisot, M. (2008). Foregrounding Differences: A performative approach to the coordination of distributed work and information infrastructures in use, Ph.D. Thesis, Departments of Informatics, University of Oslo, Norway.

Hanseth, O. (2000). The Economics of Standards, in Ciborra *et al.* (eds.), *From Control to Drift. The Dynamics of Corporate Information Infrastructures*, Oxford: Oxford University Press, pp. 56–70.

Hanseth, O. (2001). Gateways – Just as important as standards. How the internet won the 'religious war' about standards in Scandinavia, *Knowledge, Technology and Policy* **14**(3): 71–89.

Hanseth, O. and Aanestad, M. (2003). Bootstrapping Networks, Infrastructures and Communities, *Methods of Information in Medicine* **42**: 384–391.

Hanseth, O. and Ciborra, C. (2007). *Risk, Complexity and ICT*, Cheltenham, UK; Northampton, MA, USA: Edward Elgar Publishing.

Hanseth, O. and Lundberg, N. (2001). Information Infrastructure in Use – An empirical study at a radiology department, *Computer Supported Cooperative Work (CSCW). The Journal of Collaborative Computing* **10**(3–4): 347–372.

Hanseth, O. and Monteiro, E. (1997). Inscribing Behaviour in Information Infrastructure Standards, *Accounting Management and Information Technology* **7**(4): 183–211.

Hanseth, O., Monteiro, E. and Hatling, M. (1996). Developing Information Infrastructure: The tension between standardization and flexibility, *Science, Technology and Human Values* **21**(4): 407–426.

Holland, J. (1995). *Hidden Order*, Massachusetts: Addison-Wesley Reading.

Hovav, A. and Schuff, D. (2005). 'The Changing Dynamic of the Internet' Early and Late Adopters of Ipv6 Standard, *Communications of the Association of the Information Systems* **15**: 242–262.

Hughes, T.P. (1987). The Evolution of Large Technical Systems, in W.E. Bijker, T.P. Hughes and T. Pinch (eds.) *The Social Construction of Technological Systems*, Cambridge, MA: MIT Press.

Kahn, R.E. (1994). The Role of Government in the Evolution of the Internet, *Communications of the ACM* **37**(8): 415–419, Special issue on Internet technology.

Kahn, R.E. (2006). Personal interview on phone 6th December 2006.

Kallinikos, J. (2004). Deconstructing Information Packages: Organizational and behavioural implications of large scale information systems, *Information Technology and People* **17**(1): 8–30.

Kallinikos, J. (2006). Information out of Information: On the self-referential dynamics of information growth, *Information Technology and People* **19**(1): 98–115.

Kallinikos, J. (2007). Technology, Contingency and Risk: The vagaries of large-scale information systems, in O. Hanseth and C. Ciborra (eds.) *Risk, Complexity and ICT*. Cheltenham, UK; Northampton, MA: Edward Elgar Publishing, pp. 46–74.

Kayworth, T. and Sambamurthy, S. (2000). Facilitating Localized Exploitation of Enterprise Wide Integration in the use of IT Infrastructures: The role of

PC/LAN infrastructure standards, *The Data Base for Advances in Information Systems* **31**(4): 54–80.

Kling, R. (1992). Behind the Terminal: The critical role of computing infrastructure in effective information systems' development and use, in W. Cotterman and J. Senn (eds.) *Challenges and Strategies for Research in Systems Development*, London: John Wiley, pp. 153–201.

Kling, R. and Scacchi, W. (1982). The Web of Computing: Computing technology as social organization, *Advances in Computers*, New York: Academic Press, Vol. 21, pp. 3–87.

Kozlowski, S. and Klein, K. (2000). A Multi-Level Approach to Theory and Research in Organizations: Contextual, temporal and emergent processes, in K. Klein and S. Kozlowski (eds.) *Multilevel Theory, Research and Methods in Organizations*, San Francisco: Jossey-Bass, pp. 3–90.

Latour, B. (1999). *Pandora's Hope. Essays on the Reality of Science Studies*, Cambridge, MA; London, UK: Harvard University Press.

Lee, C., Dourish, P. and Mark, G. (2006). The Human Infrastructure of the Cyberinfrastructure, Proceedings of CSCW'06 (Banf, Canada); New York: ACM Press, 483–492.

Leiner, B.M., Cerf, V.C., Clark, D.D., Kahn, R.E., Kleinrock, L., Lynch, D.C., Postel, J., Roberts, L.G. and Wolff, S.S. (1997). The Past and Future History of the Internet, *Communications of the ACM* **40**(2): 102–108.

Lessig, L. (2001). *The Future of Ideas: The fate of the commons in a connected world*, New York: Random House.

Lindgren, R., Hendfridsson, O. and Schultze, U. (2004). Design Principles for Competence Management Systems: A synthesis of an action research study, *MIS Quarterly* **28**(3): 435–472.

Lyytinen, K. and Fomin, W. (2002). Achieving High Momentum in the Evolution of Wireless Infrastructures: The battle over the 1G solutions, *Telecommunications Policy* **26**: 149–190.

March, J.G. (1991). Exploration and Exploitation in Organizational Learning, *Organization Science* **2**(1): 71–78.

Markus, M.L., Majchrzak, A. and Gasser, L. (2002). A Design Theory for Systems That Support Emergent Knowledge Processes, *MIS Quarterly* **26**(3): 179–212.

Markus, M.L., Steinfield, C.W., Wigand, R.T. and Minton, G. (2006). Industry-Wide Information Systems Standardization as Collective Action: The case of the US residential mortgage industry, *MIS Quarterly* **30**(Special Issue on Standardization August 2006): 439–465.

Mashey, J. (2009). The Long Road to 64 Bits, *Communications of the ACM* **52**(1): 45–53.

Monteiro, E. (1998). Scaling Information Infrastructure: The case of the next generation IP in internet, *The Information Society* **14**(3): 229–245.

Nickerson, J.V. and zur Muehlen, M. (2006). The Ecology of Standards Processes: Insights from internet standard making, *MIS Quarterly* **30**(5): 467–488.

Olle, T.W., Soland, H.G. and Tully., C.J. (eds.) (1983). *Information Systems Design Methodologies: A feature analysis*, Amsterdam, The Netherlands: Elsevier Science Publishers B.V.

Parnas, D.L. (1972). A Technique for Software Module Specification with Examples, *Communications of the ACM* **15**(5): 330–336.

Pipek, V. and Wulf, V. (2009). Infrastructuring: Toward an integrated perspective on the design and use of information technology, *Journal of the Association for Information Systems* **10**(5): 447–473.

Porra, J. (1999). Colonial Systems, *Information Systems Research* **10**(1): 38–69.

RFC (1994). The Internet Standards Process – Revision 2, RFC 1602, IAB and IESG [www document] http://www.ietf.org/rfc/rfc1602.txt.

RFC (1995). The Recommendation for the IP Next Generation Protocol, RFC 1752, IAB and IESG [www document] http://www.ietf.org/rfc/rfc1752.txt.

Rheingold, H. (1993). *The Virtual Community: Homesteading the electronic frontier*, Reading, MA: Addison Wesley.

Robey, D. (2003). Identity, Legitimacy, and the Dominant Research Paradigm: An alternative prescription for the IS discipline, *Journal of the Association of for Information Systems* **4**(6): 352–359.

Rose, M.T. (1992). The Future of OSI: A modest prediction, in G. Neufeld and B. Plattner (eds.) Proceedings of the Usenix Conference 1992; Berkley USA: USENIX Association.

Ross, D.T. and Schoman Jr., K.E. (1977). Structured Analysis for Requirements Definition, *IEEE Transactions of Software Engineering* **SE 3**(1): 69–84.

Russell, A. (2006). Rough Consensus and Running Code' and the Internet – OSI standards war, *IEEE Annals of the History of Computing* **28**(July–September): 48–61.

Saltzer, J.H., Reed, D.P. and Clark, D.D. (1984). End-to-End Arguments in Systems Design, *ACM Transactions on Computer Systems* **2**: 277–288.

Sauer, C. and Willcocks, L. (2007). Unreasonable Expectations – NHS IT, Greek choruses and the games institutions play around mega-programmes, *Journal of Information Technology* **22**: 195–201.

Scacchi, W. (2009). Understanding Requirements for Open Source Software, in K. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson (eds.) *Design Requirements Engineering: A ten-year perspective*, LNBIP 14, Berling and Heidelberg: Springer-Verlag, pp. 467–494.

Schmidt, S.K. and Werle, R. (1998). *Coordinating Technology. Studies in the International Standardization of Telecommunications*, Cambridge MA: MIT Press.

Scott, R.W. (2001). *Institutions and Organizations*, London: Sage.

Shapiro, C. and Varian, H.R. (1999). *Information Rules: A strategic guide to the network economy*, Boston, MA: Harvard Business School Press.

Simon, H. (1969). *The Sciences of the Artificial*, Cambridge, MA: MIT Press.

Stacey, R.D. (1996). *Complexity and Creativity in Organisations*, San Francisco: Berrett-Koehler.

Star, L.S. and Ruhleder, K. (1996). Steps Toward an Ecology of Infrastructure: Design and access of large information spaces, *Information Systems Research* **7**(1): 111–134.

Stefferud, E. (1994). Paradigms Lost, *Connexions. The Interoperability Report* **8**(1).

Steinberg, S.G. (1995). Addressing the Future of the Net, *WIRED* **3**(May): 141–144.

Tilson, D. (2008). Reconfiguring to Innovate: Innovation networks during the evolution of wireless services in the United States and the United Kingdom, Ph.D. Thesis, Department of Information Systems, Case Western Reserve University.

Tilson, D. and Lyytinen, K. (2006). The 3G Transition: Changes in the US wireless industry, *Telecommunication Policy* **30**: 569–586.

Tuomi, I. (2002). *Networks of Innovation. Change and Meaning in the Age of the Internet*, Oxford, UK: Oxford University Press.

Wagner, A. (2007). *Robustness and Evolvability in Living Systems*, Princeton, NJ: Princeton University Press.

Walls, J.G., Widmeyer, G.R. and El Sawy, O.A. (1992). Building an Information System Design Theory for Vigilant EIS, *Information Systems Research* **3**(1): 36–59.

Walls, J.G., Widmeyer, G.R. and El Sawy, O.A. (2004). Assessing Information System Design Theory in Perspective: How useful was our 1992 rendition? *Journal of Information Technology Theory and application* **6**(2): 43–58.

Weick, K.E. (1989). Theory Construction as Disciplined Imagination, *Academy of Management Review* **14**(4): 516–531.

Weill, P. and Broadbent, M. (1998). *Leveraging the New Infrastructure*, Cambridge, MA: Harvard Business School Press.

Wigand, R.T., Lynne Markus, M., Steinfield, C.W. and Minton, G. (2006). Standards, Collective Action and IS Development – Vertical information systems standards in the US home mortgage industry, *MIS Quarterly*, Special Issue on Standards and Standardization **30**: 439–465.

Williams, R. and Pollock, N. (2008). *Software and Organisations – The biography of the enterprise-wide system or how SAP conquered the world*, London: Routledge.

Yoo, Y., Lyytinen, K. and Yang, E. (2005). The Role of Standards in Innovation and Diffusion of Broadband Mobile Services: The case of South Korea, *Journal of Strategic Information Systems* **14**(2): 323–353.

Zimmerman, A. (2007). A Socio-Technical Framework for Cyberinfrastructure Design, in Proceedings of e-Social Science Conference (Ann Arbor, Michigan, October).

Zittrain, J. (2006). The Generative Internet, *Harward Law Review* **119**: 1974–2040.

About the authors

Ole Hanseth is Professor in the Department of Informatics, University of Oslo. His research focuses mainly on the interplay between social and technical issues in the development and use of large-scale networking applications. He is also a visiting professor at the Department of Management, London School of Economics.

Kalle Lyytinen is Iris S. Wolstein Professor at Case Western Reserve University, USA, Adjunct Professor at University of Jyvaskyla, Finland, and a visiting professor at University

of Loughborough, UK. He serves on the editorial boards of leading information systems journals including *Journal of AIS* (Editor-in-Chief), *Journal of Strategic Information Systems, Information and Organization, Requirements Engineering Journal, Information Systems Journal, Scandinavian Journal of Information Systems, Journal of Information Technology*, and *Information Technology and People*. He is AIS Fellow (2004), and the former chairperson of IFIP WG 8.2, and a founding member of SIGSAND. He received Dr. h.c. from Umeå University, Sweden in 2008 for his work on social informatics. He has published over 200 articles and conference papers and edited or written 11 books on topics related to nature of IS discipline, system design, method engineering, digital innovation, risk assessment, computer supported cooperative work, standardization and ubiquitous computing. He is currently involved in research that looks at the IT-induced radical innovation in software development, IT innovation in architecture, engineering and construction industry, requirements discovery and modeling for large scale systems, ERP implementation processes, and the adoption of wireless services in the UK and the US.